

ADSP-2156x SHARC+ Processor Hardware Reference

Revision 1.0, December 2020

Part Number
82-100137-01



Notices

Copyright Information

© 2020 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, CrossCore, EngineerZone, EZ-Board, EZ-KIT Lite, EZ-Extender, SHARC, SHARC+, Blackfin+, A²B and VisualDSP++ are registered trademarks of Analog Devices, Inc.

EZ-KIT Mini and SigmaStudio are trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Contents

Introduction

Power and Clock Management (DPM/RCU/CGU/CDU)	1-1
System Interrupts and Triggers (SEC/TRU)	1-2
System Memory (L2CTL/DMC/OTPC/SMPU).....	1-3
Direct Memory Access (DMA/MDMA/EMDMA/CRC).....	1-4
Peripherals	1-5
General-Purpose I/O (GPIO) Peripherals.....	1-5
DAI/SRU Peripherals	1-7
System Accelerators (FIR/IIR)	1-7
Security and Protection (SPU/PKTE/PKIC/PKA/TRNG).....	1-8
Safety (TMU)	1-9
Analog Subsystem (HADC).....	1-9
System Debug (SCB/SWU/DBG/TAPC/CSPFT/STM)	1-9

Clock Generation Unit (CGU)

CGU Features	2-1
CGU Functional Description.....	2-1
ADSP-2156x CGU Register List	2-2
ADSP-2156x CGU Interrupt List	2-3
ADSP-2156x CGU Trigger List.....	2-3
CGU Definitions.....	2-3
CGU PLL Block Diagram	2-4
CLKOUT Selections	2-5
CGU Operating Modes	2-6
CGU Power-up Sequence	2-6
CGU Event Control	2-6
Oscillator Watchdog	2-7

CGU Programming Model	2-9
Configuring CGU Modes	2-9
Changing the Clock Frequencies	2-9
Changing the PLL Clock Frequency	2-9
Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency	2-10
Changing the OCLK Frequency	2-11
Selecting SCLK1_0 Source	2-11
Changing SCLK1_0 Frequencies	2-12
Aligning All Clocks	2-12
Shutting Off CCLKn from Another Master	2-13
Valid Clock Multiplier Settings	2-13
PLL Bypass and PLL Disable	2-13
ADSP-2156x Specific Information	2-14
ADSP-2156x CGU Register Descriptions	2-14
Core Clock Buffer Disable Register	2-16
Core Clock Buffer Status Register	2-17
CLKOUT Select Register	2-18
Control Register	2-20
Clocks Divisor Register	2-22
DIV Register Extension	2-25
Oscillator Watchdog Register	2-26
PLL Control Register	2-28
Revision ID Register	2-30
System Clock Buffer Disable Register	2-31
System Clock Buffer Status Register	2-33
Status Register	2-35
Time Stamp Counter 32 LSB Register	2-38
Time Stamp Counter 32 MSB Register	2-39
Time Stamp Control Register	2-40
Time Stamp Counter Initial 32 LSB Value Register	2-41
Time Stamp Counter Initial MSB Value Register	2-42

Clock Distribution Unit (CDU)

CDU Features.....	3-1
CDU Functional Description.....	3-1
CDU Block Diagram.....	3-1
CDU Definitions.....	3-2
CDU Clock Configuration Options	3-3
CDU Programming Model.....	3-3
Changing the PLL and Clock Frequency	3-3
Changing the Clock Frequency.....	3-4
ADSP-2156x CDU Register Descriptions	3-4
CDU Configuration	3-6
CLKIN Select	3-7
CDU Revision ID	3-8
CDU Status	3-9

Dynamic Power Management (DPM)

DPM Features.....	4-1
DPM Functional Description	4-1
ADSP-2156x DPM Register List.....	4-1
DPM Definitions	4-1
DPM Operating Modes.....	4-2
Reset State.....	4-2
Full-on Mode.....	4-2
DPM Event Control	4-3
DPM Programming Model.....	4-3
ADSP-2156x DPM Register Descriptions	4-4
Control Register	4-5
Peripherals Disable Register 0	4-6
Peripherals Disable Register 1	4-8
Revision ID	4-10

Status Register	4-11
-----------------------	------

Reset Control Unit (RCU)

RCU Features	5-1
RCU Functional Description	5-1
ADSP-2156x RCU Register List.....	5-2
ADSP-2156x RCU Trigger List	5-3
RCU Definitions	5-3
RCU Architectural Concepts	5-3
RCU Status and Error Signals.....	5-4
ADSP-2156x RCU Register Descriptions	5-4
Boot Code Register	5-6
Core Reset Outputs Control Register	5-9
Core Reset Outputs Status Register	5-10
Control Register	5-11
Message Register	5-13
Message Clear Bits Register	5-16
Message Set Bits Register	5-17
System Interface Disable Register	5-18
System Interface Status Register	5-19
System Reset Request Status Register	5-20
Status Register	5-21
Software Vector Register 0	5-23
SVECT Lock Register	5-24

System Event Controller (SEC)

SEC Features.....	6-1
SEC Functional Description	6-2
ADSP-2156x SEC Register List.....	6-2
ADSP-2156x SEC Interrupt List	6-3
ADSP-2156x SEC Trigger List	6-3

ADSP-2156x Interrupt List	6-3
SEC Definitions	6-10
SEC Block Diagram.....	6-11
SEC Fault Interface (SFI)	6-11
SEC Core Interface (SCI)	6-12
SEC Source Interface (SSI)	6-13
SEC Architectural Concepts	6-13
System Interrupt Acknowledge.....	6-13
System Interrupt Groups.....	6-14
System Interrupt Flow.....	6-14
System Interrupt Priorities	6-15
SEC Error.....	6-15
SEC Programming Model.....	6-15
Programming Concepts	6-16
Programming Examples.....	6-16
Fault Management Interface Programming Model	6-16
Configuring a System Source to Interrupt a Core.....	6-17
Core/SEC Handshake Requirements to Ensure Proper Interrupt Handling	6-18
Configuring a System Source as a Fault.....	6-19
SEC Programming Restrictions	6-19
ADSP-2156x SEC Register Descriptions	6-19
SCI Active Register n	6-21
SCI Control Register n	6-22
SCI Group Mask Register n	6-24
SCI Priority Level Register n	6-26
SCI Priority Mask Register n	6-28
Core Pending Register n	6-29
SCI Source ID Register n	6-30
SCI Status Register n	6-31
Global End Register	6-33
Fault COP Period Register	6-34

Fault COP Period Current Register	6-35
Fault Control Register	6-36
Fault Delay Register	6-39
Fault Delay Current Register	6-40
Fault End Register	6-41
Fault Source ID Register	6-42
Fault System Reset Delay Register	6-43
Fault System Reset Delay Current Register	6-44
Fault Status Register	6-45
Global Control Register	6-47
Global Status Register	6-48
Global Raise Register	6-50
Source Control Register n	6-51
Source Status Register n	6-54

Trigger Routing Unit (TRU)

TRU Features	7-1
TRU Functional Description	7-1
ADSP-2156x TRU Register List.....	7-2
ADSP-2156x TRU Interrupt List	7-2
ADSP-2156x TRU Trigger List	7-2
TRU Definitions	7-3
TRU Block Diagram	7-3
TRU Architectural Concepts	7-4
TRU Programming Model.....	7-4
Programming Concepts.....	7-4
Programming Examples.....	7-5
Configuring a Simple Trigger Sequence.....	7-5
TRU Event Control	7-5
TRU Status and Error Signals.....	7-5

ADSP-2156x TRU Register Descriptions	7-5
Error Address Register	7-6
Global Control Register	7-7
Master Trigger Register	7-9
Slave Select Register	7-10
Status Information Register	7-11
L2 System Memory	
L2 System Memory Features	8-1
L2 System Memory Functional Description.....	8-1
ADSP-2156x L2CTL Register List	8-1
ADSP-2156x L2CTL Interrupt List	8-2
ADSP-2156x L2CTL Trigger List	8-3
L2 System Memory Block Diagram.....	8-3
L2 System Memory Architectural Concepts.....	8-3
Access Characteristics	8-4
Read/Write Latency and Throughput	8-4
L2 Memory Controller Block Diagram (Instance)	8-4
Arbitration and Priority.....	8-5
Data Integrity.....	8-6
ECC Algorithm.....	8-6
ECC Hardware Control	8-8
ECC Error Management	8-8
Memory Refresh.....	8-9
Power Modes.....	8-9
L2 System Memory Event Control.....	8-10
ECC Error Interrupt.....	8-10
ADSP-2156x L2CTL Register Descriptions	8-10
Control Register	8-12
Error Type 0 Address Register	8-15
Error Type 1 Address Register	8-16

ECC Error Address 0 Register	8-17
ECC Error Address 1 Register	8-18
ECC Error Address 2 Register	8-19
ECC Error Address 3 Register	8-20
ECC Error Address 4 Register	8-21
ECC Error Address 5 Register	8-22
ECC Error Address 6 Register	8-23
ECC Error Address 7 Register	8-24
ECC Error Address 8 Register	8-25
Error Type 0 Register	8-26
Error Type 1 Register	8-27
Initialization Register	8-28
Initialization Status Register	8-30
Power Control Register	8-32
Revision ID Register	8-34
Read Priority Count Register	8-35
Scrub Start Address Register	8-36
Scrub Count Register	8-37
Scrub Control Register	8-38
Status Register	8-40
Write Priority Count Register	8-43

Dynamic Memory Controller (DMC)

DMC Features	9-1
Feature Exclusions	9-2
DMC Functional Description	9-2
ADSP-2156x DMC Register List	9-3
ADSP-2156x DMC Register List	9-3
Protocol Controller.....	9-4
Efficiency Controller	9-4

Page-Based Scheduling	9-4
Same Master Transaction Scheduling	9-5
DMC Read Data Buffer	9-5
Closed Page Per Bank.....	9-5
SCB ID-Based Priority	9-5
Delaying up to Eight Auto-Refresh Commands.....	9-6
Page and Bank Interleaving	9-6
System Crossbar Slave Interface.....	9-6
Read/Write Command and Data Buffers.....	9-7
Peripheral Bus Slave Interface	9-7
Architectural Concepts	9-7
Controller On Die Termination (ODT).....	9-8
Mode Register Set and Extended Mode Register Set Command.....	9-8
DDR3 Reset Functionality.....	9-8
DDR3 SDRAM Organization.....	9-8
DMC Clocking	9-9
DMC DMA	9-9
DMC Operating Modes	9-10
Self-Refresh Mode	9-10
DMC Event Control.....	9-10
DMC Programming Model	9-10
PHY DLL Calibration	9-11
DDR3 ZQ Calibration Short CMD	9-11
DDR3 ZQ Calibration Long CMD.....	9-11
Leveling Techniques	9-12
Initializing the DMC	9-13
Resetting the DMC Lane	9-13
Performing ZQ Calibration	9-13
Programming the DMC Controller	9-14
ADSP-2156x DMC Register Descriptions	9-15
Configuration Register	9-17

Control Register	9-19
DLL Control Register	9-23
Data Calibration Address Register	9-24
Data Calibration Data 0 Register	9-25
Data Calibration Data 1 Register	9-26
Efficiency Control Register	9-27
Shadow EMR3 Register	9-31
Shadow MR0 Register (DDR3)	9-32
Shadow MR1 Register (DDR3)	9-34
Shadow MR2 Register (DDR3)	9-37
Mask (Mode Register Shadow) Register	9-39
Priority ID Register 1	9-41
Priority ID Register 2	9-42
Priority ID Mask Register 1	9-43
Priority ID Mask Register 2	9-44
DMC Read Data Buffer ID Register 1	9-45
DMC Read Data Buffer ID Register 2	9-46
DMC Read Data Buffer Mask Register 1	9-47
DMC Read Data Buffer Mask Register 2	9-48
Status Register	9-49
Timing 0 Register	9-52
Timing 1 Register	9-54
Timing 2 Register	9-55
ADSP-2156x DMC Register Descriptions	9-56
DDR CA Lane Control Register	9-57
Data Lane 0 Control Register 0	9-59
Data Lane 0 Control Register 1	9-61
Data Lane 1 Control Register 0	9-62
Data Lane 1 Control Register 1	9-64
DDR ROOT Module Control Register	9-65

DDR Calibration Control Register 0	9-67
DDR Calibration Control Register 1	9-68
DDR Calibration Control Register 2	9-69

One-Time Programmable Memory Controller (OTPC)

OTPC Features	10-1
OTPC Functional Description	10-1
ADSP-2156x OTPC Register List	10-1
ADSP-2156x OTPC Interrupt List	10-2
Error Correction	10-2
OTP Layout	10-2
OTPC Event Control	10-5
OTPC Interrupt Signals	10-5
OTPC Status and Error Signals	10-5
OTP API Overview	10-5
OTP Programming	10-5
OTP Program	10-5
OTP Reading	10-6
OTP Get Field	10-7
OTP Counters	10-7
Lock API	10-8
ADSP-2156x OTPC Register Descriptions	10-8
OTP Security State Register	10-9
OTP Status Register	10-10

System Memory Protection Unit (SMPU)

SMPU Features	11-1
SMPU Functional Description	11-2
ADSP-2156x SMPU Register List	11-3
SMPU Interrupts	11-3
Memory Writes	11-4

Memory Reads	11-4
ID Comparison	11-4
Memory Region.....	11-7
SMPU Definitions	11-8
SMPU Block Diagram.....	11-9
SMPU Architectural Concepts	11-9
SMPU Operating Modes	11-10
SMPU Interrupt Signals	11-10
SMPU Status and Error Signals	11-11
ADSP-2156x SMPU Register Descriptions	11-11
Bus Error Address Register	11-13
Bus Error Details Register	11-14
SMPU Control Register	11-15
Interrupt Address Register	11-17
Interrupt Details Register	11-18
Region n Address Register	11-19
Region n Control Register	11-20
SMPU Revision ID Register	11-23
Region n ID A Register	11-24
Region n ID B Register	11-25
Region n ID Mask A Register	11-26
Region n ID Mask B Register	11-27
SMPU Control Secure Accesses Register	11-28
Region n Control Secure Accesses Register	11-30
SMPU Status Register	11-31
General-Purpose Ports (PORT)	
PORT Features	12-2
PORT Functional Description.....	12-2
ADSP-2156x PORT Register List	12-3

ADSP-2156x PORT Trigger List.....	12-3
ADSP-2156x PINT Register List	12-4
ADSP-2156x PINT Interrupt List	12-4
ADSP-2156x PINT Trigger List.....	12-5
ADSP-2156x PADS Register List	12-5
PORT Architectural Concepts.....	12-6
Internal Interfaces	12-6
External Interfaces.....	12-6
GPIO Pin Function.....	12-6
Input Mode.....	12-6
Output Mode	12-6
Trigger Toggle Mode.....	12-7
Open-Drain Mode	12-7
Port Multiplexing Control.....	12-7
PORT Event Control.....	12-8
PORT Interrupt Signals	12-8
PORT Programming Model	12-10
ADSP-2156x PORT Register Descriptions	12-13
Port x GPIO Data Register	12-15
Port x GPIO Data Clear Register	12-19
Port x GPIO Data Set Register	12-23
Port x GPIO Output Toggle Register	12-26
Port x GPIO Direction Register	12-29
Port x GPIO Direction Clear Register	12-33
Port x GPIO Direction Set Register	12-37
Port x Function Enable Register	12-40
Port x Function Enable Clear Register	12-43
Port x Function Enable Set Register	12-46
Port x GPIO Input Enable Register	12-49
Port x GPIO Input Enable Clear Register	12-52

Port x GPIO Input Enable Set Register	12-55
Port x GPIO Lock Register	12-58
Port x Multiplexer Control Register	12-60
Port x GPIO Polarity Invert Register	12-62
Port x GPIO Polarity Invert Clear Register	12-66
Port x GPIO Polarity Invert Set Register	12-69
Port x GPIO Trigger Toggle Register	12-72
ADSP-2156x PINT Register Descriptions	12-73
PINT Assign Register	12-75
PINT Edge Clear Register	12-77
PINT Edge Set Register	12-80
PINT Invert Clear Register	12-83
PINT Invert Set Register	12-86
PINT Latch Register	12-89
PINT Mask Clear Register	12-93
PINT Mask Set Register	12-96
PINT Pin State Register	12-99
PINT Request Register	12-103
ADSP-2156x PADS Register Descriptions	12-107
DAI0 1 to 10 pins DS control	12-108
DAI0 11 to 20 pins DS control	12-111
DAI0 Port Input Enable Control Register	12-113
DAI1 1 to 10 pins DS control	12-114
DAI1 11 to 20 pins DS control	12-117
DAI1 Port Input Enable Control Register	12-119
DAIx Pull-Down Enable	12-120
DAIx Pull-Up Enable	12-121
Non-GPIO Drive Strength Register	12-122
Peripheral PAD Configuration0 Register	12-124
PORTA 0 to 7 pins DS control	12-125

PORTA 8 - 15 pins DS control	12-128
PORTB 0 to 7 pins DS control	12-131
PORTB 8 - 15 pins DS control	12-134
PORTC 0 to 7 pins DS control	12-137
PORTx Pull-Down Enable	12-140
PORTx Pull-Up Enable	12-141

Watchdog Timer (WDOG)

WDOG Features.....	13-1
WDOG Functional Description	13-1
ADSP-2156x WDOG Register List.....	13-2
ADSP-2156x WDOG Interrupt List	13-3
WDOG Block Diagram.....	13-3
Internal Interface	13-3
External Interface	13-3
ADSP-2156x WDOG Register Descriptions	13-3
Count Register	13-5
Control Register	13-6
Watchdog Timer Status Register	13-7
Watchdog Timer Window Register	13-9

Link Port (LP)

LP Features	14-1
LP Functional Description.....	14-1
ADSP-2156x LP Register List	14-2
ADSP-2156x LP Interrupt List	14-2
ADSP-2156x LP Trigger List.....	14-2
ADSP-2156x LP DMA Channel List.....	14-3
Block Diagram.....	14-3
External Connections	14-4
Internal Blocks	14-4

Architectural Concepts	14-4
Link Port Protocol.....	14-4
FIFO Buffers	14-7
Handshake for Link Port Enable Process	14-9
Clocking	14-10
Multi-Processor Connectivity	14-10
LP Operating Modes	14-12
LP Data Transfer Modes.....	14-12
Core Data Transfers	14-12
DMA Data Transfers.....	14-12
LP Event Control.....	14-13
Interrupt Signals	14-13
Enabling Link Port Interrupts	14-13
Status and Error Signals.....	14-14
LP Programming Model	14-14
Setting Up a DMA Transmit Operation	14-14
Setting Up a DMA Receive Operation.....	14-15
Setting Up a Core Transmit Operation.....	14-16
Setting Up a Core Receive Operation	14-16
ADSP-2156x LP Register Descriptions	14-17
Control Register	14-18
Clock Divider Value Register	14-20
Receive Buffer Register	14-21
Status Register	14-22
Transmit Buffer Register	14-24
Shadow Input Transmit Buffer Register	14-25
Shadow Output Transmit Buffer Register	14-26

Serial Peripheral Interface (SPI)

SPI Features.....	15-1
SPI Functional Description.....	15-2

ADSP-2156x SPI Register List	15-2
ADSP-2156x SPI Interrupt List	15-3
ADSP-2156x SPI Trigger List.....	15-4
ADSP-2156x SPI DMA Channel List	15-5
SPI Block Diagram	15-5
Transfer Protocol	15-6
Clock Considerations	15-7
Controlling Delay Between Frames	15-8
Flow Control	15-9
Slave Select Operation	15-10
Beginning and Ending a Non-DMA SPI Transfer	15-11
Transmit Operation in Non-DMA Mode	15-12
Receive Operation in Non-DMA Mode	15-12
Dual I/O Mode	15-12
Quad I/O Mode (SPI1/SPI2 only)	15-13
Fast Mode	15-14
Memory-Mapped Mode (SPI2 only).....	15-15
Memory-Mapped Description of Operation.....	15-16
Memory-Mapped Architectural Concepts.....	15-17
Memory-Mapped Read Accesses.....	15-19
Memory-Mapped High-Performance Features.....	15-22
Merged Read Accesses	15-22
Wrap Around Accesses	15-22
Execute-In-Place (XIP, SPI2 only)	15-23
Memory-Mapped Mode Error Status Bits	15-24
Memory-Mapped Programming Guidelines	15-25
SPI Interrupt Signals	15-27
Data Interrupts.....	15-27
Status Interrupts.....	15-28
Error Conditions	15-28

SPI Programming Concepts.....	15–29
Master Operation in Non-DMA Modes	15–30
Slave Operation in Non-DMA Modes	15–30
Configuring DMA Master Mode.....	15–31
Configuring DMA Slave Mode Operation.....	15–32
ADSP-2156x SPI Register Descriptions	15–33
Clock Rate Register	15–35
Control Register	15–36
Delay Register	15–42
Masked Interrupt Condition Register	15–43
Masked Interrupt Clear Register	15–45
Interrupt Mask Register	15–48
Interrupt Mask Clear Register	15–50
Interrupt Mask Set Register	15–53
Memory Mapped Read Header	15–56
SPI Memory Top Address	15–60
Receive FIFO Data Register	15–61
Received Word Count Register	15–62
Received Word Count Reload Register	15–63
Receive Control Register	15–64
Slave Select Register	15–67
Status Register	15–70
Transmit FIFO Data Register	15–75
Transmitted Word Count Register	15–76
Transmitted Word Count Reload Register	15–77
Transmit Control Register	15–78
Octal Serial Peripheral Interface (OSPI)	
OSPI Features.....	16–1
OSPI Functional Description	16–2

ADSP-2156x OSPI Register List	16-2
ADSP-2156x OSPI Interrupt List	16-3
OSPI Block Diagram.....	16-3
Architectural Concepts	16-4
Direct Access Controller (DAC)	16-4
Software Triggered Instruction Generator (STIG)	16-6
Servicing STIG Request	16-7
Arbitration Between DAC and STIG Access	16-7
Auto Polling for DAC Write Access	16-7
SPI Command Translation.....	16-8
Hold and Reset Control	16-8
Flash Instruction Type Support	16-9
Dual Data Rate (DDR) Operations	16-9
Data Sampling	16-13
Programming Concepts	16-14
Configuring OSPI after Reset.....	16-14
Programming Dummy Cycles.....	16-15
Configuring OSPI for Optimal Use.....	16-15
Configuring OSPI for DAC Read Operation.....	16-16
Configuring OSPI for DAC Write Operation.....	16-16
Issuing STIG Command	16-17
Entering XIP mode.....	16-19
 Universal Asynchronous Receiver/Transmitter (UART)	
UART Features	17-1
UART Functional Description	17-2
ADSP-2156x UART Register List	17-3
ADSP-2156x UART Interrupt List	17-3
ADSP-2156x UART Trigger List.....	17-4
ADSP-2156x UART DMA Channel List.....	17-5
UART Block Diagram	17-5

UART Architectural Concepts	17-5
Internal Interface.....	17-5
External Interface	17-6
Hardware Flow Control.....	17-6
Bit Rate Generation	17-7
Autobaud Detection	17-7
UART Debug Features	17-9
UART Operating Modes.....	17-9
UART Mode.....	17-10
IrDA SIR Mode.....	17-10
Multi-Drop Bus Mode.....	17-10
UART Data Transfer Modes.....	17-12
UART Mode Transmit Operation (Core)	17-12
UART Mode LIN Break Command	17-12
UART Mode Receive Operation (Core).....	17-13
IrDA Transmit Operation	17-14
IrDA Receive Operation	17-14
MDB Transmit Operation.....	17-15
MDB Receive Operation	17-16
DMA Mode.....	17-16
Mixing DMA and Core Modes.....	17-17
Setting Up Hardware Flow Control.....	17-17
UART Event Control.....	17-18
Interrupt Masks	17-18
Interrupt Servicing	17-18
Transmit Interrupts	17-19
Receive Interrupts.....	17-20
Status Interrupts.....	17-21
Multi-Drop Bus Events.....	17-22
UART Programming Model	17-22
Detecting Autobaud	17-22

Using Common Initialization Steps.....	17-23
Using Core Transfers	17-23
Using DMA Transfers.....	17-23
Using Interrupts	17-23
Setting Up Hardware Flow Control.....	17-23
ADSP-2156x UART Register Descriptions	17-24
Clock Rate Register	17-25
Control Register	17-26
Interrupt Mask Register	17-32
Interrupt Mask Clear Register	17-36
Interrupt Mask Set Register	17-38
Receive Buffer Register	17-40
Receive Shift Register	17-41
Receive Counter Register	17-42
Scratch Register	17-43
Status Register	17-44
Transmit Address/Insert Pulse Register	17-49
Transmit Hold Register	17-50
Transmit Shift Register	17-51
Transmit Counter Register	17-52
General-Purpose Timer (TIMER)	
GP Timer Features.....	18-1
ADSP-2156x TIMER Register List	18-2
ADSP-2156x TIMER Interrupt List	18-2
ADSP-2156x TIMER Trigger List.....	18-3
Timer Block Diagram.....	18-4
Internal Interface	18-5
Internal Timer Connections	18-5
External Interface	18-6

GP Timer Operating Modes	18-6
General Operation.....	18-6
Period, Width and Delay Register Interaction	18-7
Single-Pulse PWMOUT Mode	18-8
Continuous PWMOUT Mode	18-8
Width Capture (WIDCAP) Mode.....	18-10
Width Capture Mode Overflow.....	18-12
Windowed Watchdog (WATCHDOG) Modes	18-14
Windowed Watchdog Width Mode	18-14
Windowed Watchdog Period Mode.....	18-16
Pin Interrupt (PININT) Mode.....	18-18
External Clock (EXTCLK) Mode	18-18
GP Timer Programming Concepts	18-19
Setting Up Constantly Changing Timer Conditions	18-19
Configuring, Enabling, and Disabling One or More Timers.....	18-20
Configuring Timer Data and Status Interrupts	18-20
Configuring the Timer as a Trigger Slave.....	18-20
Ordered Trigger Toggle Mode	18-21
Using the Timer Broadcast Feature.....	18-21
Timer Illegal States	18-22
Continuous PWMOUT Mode	18-22
Single Pulse PWMOUT Mode.....	18-23
WIDCAP Mode	18-24
EXTCLK Mode	18-24
WATCHDOG Events.....	18-25
ADSP-2156x TIMER Register Descriptions	18-26
Broadcast Delay Register	18-27
Broadcast Period Register	18-28
Broadcast Width Register	18-29
Data Interrupt Latch Register	18-30

Data Interrupt Mask Register	18–31
Error Type Status Register	18–32
Run Register	18–35
Run Clear Register	18–36
Run Set Register	18–37
Status Interrupt Latch Register	18–38
Status Interrupt Mask Register	18–39
Stop Configuration Register	18–40
Stop Configuration Clear Register	18–41
Stop Configuration Set Register	18–42
Timer n Configuration Register	18–43
Timer n Counter Register	18–48
Timer n Delay Register	18–49
Timer n Period Register	18–50
Timer n Width Register	18–51
Trigger Slave Enable Register	18–52
Trigger Master Mask Register	18–53

General-Purpose Counter (CNT)

GP Counter Features	19–1
GP Counter Functional Description	19–1
ADSP-2156x CNT Register List	19–2
ADSP-2156x CNT Interrupt List	19–2
ADSP-2156x CNT Trigger List.....	19–3
GP Counter Operating Modes.....	19–3
Quadrature Encoder Mode	19–3
Binary Encoder Mode.....	19–4
Up/Down Counter Mode.....	19–4
Direction Counter Mode	19–5
Timed Direction Mode.....	19–5

GP Counter Programming Model	19-5
GP Counter General Programming Flow	19-5
GP Counter Mode Configuration.....	19-5
Configuring GP Counter Push-Button Operation	19-6
Configuring Zero-Marker-Zeros-Counter Mode	19-6
Configuring Zero-Marker-Error Mode	19-6
Configuring Zero-Once Mode.....	19-6
Configuring Boundary Auto-Extend Mode	19-7
Configuring Boundary Capture Mode.....	19-7
Configuring Boundary Compare and Boundary Zero Modes	19-7
Configuring GP Counter Push-Button Operation	19-8
GP Counter Programming Concepts.....	19-8
CNT Input Noise Filtering	19-8
Capturing Counter Interval and CNT_CNTR Read Timing	19-9
Capturing Time Interval Between Successive Counter Events	19-10
GP Counter Event Control.....	19-11
Illegal Gray and Binary Code Events	19-11
Up/Down Count Events	19-11
Zero-Count Events	19-12
Overflow Events	19-12
Boundary Match Events	19-12
Zero Marker Events	19-12
ADSP-2156x CNT Register Descriptions	19-12
Configuration Register	19-14
Command Register	19-17
Counter Register	19-20
Debounce Register	19-21
Interrupt Mask Register	19-23
Maximum Count Register	19-26
Minimum Count Register	19-27
Status Register	19-28

Media Local Bus (MLB)

Features.....	20-1
MLB Definitions	20-2
Clocking	20-3
Functional Description	20-3
ADSP-2156x MLB Register List.....	20-3
ADSP-2156x MLB Interrupt List	20-4
MediaLB Protocol	20-5
MLB Architectural Concepts	20-6
MediaLB Block Diagram	20-6
MediaLB Interface.....	20-6
Routing Fabric.....	20-7
Data Buffer RAM.....	20-7
Channel Table RAM	20-7
Address Mapping	20-7
Channel Allocation Table.....	20-8
Channel Set Up.....	20-10
Channel Descriptor Tables.....	20-10
AHB Descriptor Table (ADT).....	20-15
Interrupt Interface Block	20-19
Operating Modes	20-19
Isochronous Data Exchange.....	20-19
Asynchronous and Control Data Exchange.....	20-21
Synchronous Data Exchange.....	20-21
Data Transfer	20-22
DMA.....	20-22
Programming Model.....	20-22
Channel Initialization.....	20-22
Configure the Hardware.....	20-22
Program the CAT and the CDT.....	20-22

Program the ADT	20–24
Service	20–25
Servicing the DMA Channel Interrupts.....	20–25
Servicing the MediaLB Status Interrupts.....	20–25
Polling for MediaLB System Commands.....	20–26
ADSP-2156x MLB Register Descriptions	20–26
Peripheral Channel Mask 0 Register	20–28
Peripheral Channel Mask 1 Register	20–29
Peripheral Channel Status 0 Register	20–30
Peripheral Channel Status 1 Register	20–31
Bus Control Register	20–32
MediaLB Control 0 Register	20–34
Control 1 Register	20–36
MLB Global Control Register	20–37
HBI Channel Busy 0 Register	20–38
HBI Channel Busy 1 Register	20–39
HBI Channel Error 0 Register	20–40
HBI Channel Error 1 Register	20–41
HBI Channel Mask 0 Register	20–42
HBI Channel Mask 1 Register	20–43
HBI Control Register	20–44
Memory Interface Address Register	20–45
Memory Interface Control Register	20–46
Memory Interface Control Data 0 Register	20–47
Memory Interface Control Data 1 Register	20–48
Memory Interface Control Data 2 Register	20–49
Memory Interface Control Data 3 Register	20–50
Memory Interface Control Data Write Enable 0 Register	20–51
Memory Interface Control Data Write Enable 1 Register	20–52
Memory Interface Control Data Write Enable 2 Register	20–53

Memory Interface Control Data Write Enable 3 Register	20-54
Interrupt Enable Register	20-55
Channel Status 0 Register	20-58
Channel Status 1 Register	20-59
System Data Register	20-60
System Status Register	20-61

Two-Wire Interface (TWI)

TWI Features.....	21-1
TWI Functional Description	21-2
ADSP-2156x TWI Register List.....	21-2
ADSP-2156x TWI Interrupt List	21-3
TWI Block Diagram.....	21-3
External Interface	21-3
Serial Clock Signal (SCL).....	21-4
Serial Data Signal (SDA).....	21-4
Internal Interface.....	21-5
TWI Architectural Concepts	21-5
TWI Protocol.....	21-5
Clock Generation and Synchronization	21-6
Bus Arbitration	21-6
Start and Stop Conditions	21-7
General Call Support.....	21-7
Fast Mode	21-8
TWI Operating Modes	21-8
Repeated Start	21-8
Transmit Receive Repeated Start.....	21-8
Receive Transmit Repeated Start.....	21-9
Clock Stretching	21-10
Clock Stretching During FIFO Underflow	21-10
Clock Stretching During FIFO Overflow	21-11

Clock Stretching During Repeated Start.....	21-11
TWI Programming Model.....	21-12
General Setup.....	21-12
Slave Mode.....	21-13
Master Mode Program Flow.....	21-14
Master Mode Clock Setup.....	21-15
Master Mode Transmit.....	21-15
Master Mode Receive.....	21-16
ADSP-2156x TWI Register Descriptions	21-17
SCL Clock Divider Register	21-18
Control Register	21-19
FIFO Control Register	21-21
FIFO Status Register	21-23
Interrupt Mask Register	21-24
Interrupt Status Register	21-26
Master Mode Address Register	21-29
Master Mode Control Registers	21-30
Master Mode Status Register	21-33
Rx Data Double-Byte Register	21-36
Rx Data Single-Byte Register	21-37
Slave Mode Address Register	21-38
Slave Mode Control Register	21-39
Slave Mode Status Register	21-41
Tx Data Double-Byte Register	21-42
Tx Data Single-Byte Register	21-43
Digital Audio Interface (DAI)	
SRU Features.....	22-1
Functional Description.....	22-2
ADSP-2156x DAI Register List.....	22-2

ADSP-2156x DAI Interrupt List	22-4
ADSP-2156x DAI Trigger List	22-4
DAI Block Diagram.....	22-5
DAI Signal Naming Conventions	22-6
I/O Pin Buffers.....	22-6
Pin Buffer Signals.....	22-7
Pin Buffer Input Signal	22-7
Pin Buffer Enable Signal	22-7
Pin Buffer Functions	22-7
Pin Buffers as Signal Input.....	22-7
Pin Buffers As Signal Output.....	22-8
DAI Pin Buffer Status	22-8
DAIn Peripherals.....	22-9
Output Signals With Pin Buffer Enable Control.....	22-9
Output Signals Without Pin Buffer Enable Control.....	22-9
Signal Routing Units (SRUs)	22-9
Cross Mode Connections	22-9
Signal Routing Matrix by Groups.....	22-11
DAI Group Routing.....	22-12
Rules for SRU Connections.....	22-13
Miscellaneous Buffers and Functions.....	22-14
DAI Routing Capabilities.....	22-14
DAI Default Routing	22-16
Unused DAI Connections.....	22-18
DAI Operating Modes.....	22-18
DAI Pin Buffer Polarity.....	22-19
DAI Miscellaneous Buffer Polarity	22-19
DAI System Interrupt Controller (SIC)	22-19
Signal Routing Unit Effect Latency.....	22-22
DAI Programming Model.....	22-22
Debug Features.....	22-22

DAI Sources Overview.....	22–22
DAI0 Group A – Clock Routing Source Signals	22–23
DAI0 Group B – Serial Data Source Signals	22–25
DAI0 Group C – Frame Sync Source Signals	22–26
DAI0 Group D – Pin Signal Assignment Source Signals	22–27
DAI0 Group E – Miscellaneous Source Signals	22–30
DAI0 Group F – Pin Output Enable Source Signals	22–33
DAI1 Group A – Clock Routing Source Signals	22–34
DAI1 Group B – Serial Data Source Signals	22–35
DAI1 Group C – Frame Sync Source Signals	22–37
DAI1 Group D – Pin Signal Assignment Source Signals	22–38
DAI1 Group E – Miscellaneous Source Signals	22–41
DAI1 Group F – Pin Output Enable Source Signals	22–42
DAI Destination Registers Overview	22–44
SPORT Grouping with DAI.....	22–50
ADSP-2156x DAI Register Descriptions	22–51
Clock Routing Control Register 0	22–53
Clock Routing Control Register 1	22–55
Clock Routing Control Register 2	22–57
Clock Routing Control Register 3	22–58
Clock Routing Control Register 4	22–59
Clock Routing Control Register 5	22–60
Serial Data Routing Control Register 0	22–61
Serial Data Routing Control Register 1	22–62
Serial Data Routing Control Register 2	22–63
Serial Data Routing Control Register 3	22–64
Serial Data Routing Control Register 4	22–65
Serial Data Routing Control Register 5	22–66
Serial Data Routing Control Register 6	22–67
Frame Sync Routing Control Register 0	22–68

Frame Sync Routing Control Register 1	22-70
Frame Sync Routing Control Register 2	22-72
Frame Sync Routing Control Register 4	22-73
Global SPORT Interrupt Grouping Register	22-74
Global SPORT Enable Register	22-77
Falling-Edge Interrupt Mask Register	22-80
Core Interrupt Priority Assignment Register	22-83
Rising-Edge Interrupt Mask Register	22-86
High Priority Interrupt Latch Register	22-89
Shadow High Priority Interrupt Latch Register	22-92
Low Priority Interrupt Latch Register	22-96
Shadow Low Priority Interrupt Latch Register	22-99
Miscellaneous Control Register 0	22-103
Miscellaneous Control Register 1	22-105
Pin Buffer Enable Register 0	22-107
Pin Buffer Enable Register 1	22-108
Pin Buffer Enable Register 2	22-109
Pin Buffer Enable Register 3	22-110
Pin Buffer Assignment Register 0	22-111
Pin Buffer Assignment Register 1	22-112
Pin Buffer Assignment Register 2	22-113
Pin Buffer Assignment Register 4	22-114
Pin Status Register	22-115

Serial Port (SPORT)

Features.....	23-1
Signal Descriptions	23-3
SRU Programming	23-6
Functional Description	23-6
ADSP-2156x SPORT Register List	23-6

ADSP-2156x SPORT Interrupt List	23-7
ADSP-2156x SPORT Trigger List.....	23-9
ADSP-2156x SPORT DMA Channel List.....	23-10
Block Diagram.....	23-11
Architectural Concepts	23-12
Multiplexer Logic	23-13
Data Types and Companding	23-16
Companding as a Function.....	23-17
Transmit Path.....	23-17
Receive Path	23-19
Operating Modes and Options	23-19
Serial Word Length.....	23-21
Clock Sample and Drive Edges	23-21
Frame Sync Options	23-23
Data-Dependent versus Data-Independent Frame Syncs	23-23
Support for Edge-Detected and Level-Sensitive Frame Syncs.....	23-23
Early versus Late Frame Syncs	23-24
Framed versus Unframed Frame Syncs	23-25
Frame Sync Polarity.....	23-26
Premature Frame Sync Error Detection	23-26
Mode Selection	23-27
Standard DSP Serial Mode.....	23-27
Stereo Modes.....	23-28
I ² S Mode.....	23-29
Left-Justified Mode	23-30
Right-Justified Mode.....	23-31
Multichannel (TDM) Mode.....	23-33
Packed I ² S Mode.....	23-36
Gated Clock Mode	23-37
Data Transfers and Interrupts	23-38
Data Buffers	23-38

Data Buffer Status	23–40
Single-Word (Core) Transfers	23–40
DMA Transfers.....	23–41
Data Transfer Interrupt	23–42
Error Detection (Status) Interrupt	23–43
Grouping of SPORTs.....	23–44
SPORT Programming Model	23–46
Initializing Core-Driven (Non-MCM) Transfers.....	23–46
Initializing Multichannel Transfers	23–48
Using DMA for SPORT Transfers.....	23–49
Using Companding as a Function.....	23–49
Programming Global SPORT Groups	23–50
Disabling Global SPORT Groups.....	23–51
ADSP-2156x SPORT Register Descriptions	23–51
Half SPORT 'A' Multichannel 0-31 Select Register	23–53
Half SPORT 'B' Multichannel 0-31 Select Register	23–54
Half SPORT 'A' Multichannel 32-63 Select Register	23–55
Half SPORT 'B' Multichannel 32-63 Select Register	23–56
Half SPORT 'A' Multichannel 64-95 Select Register	23–57
Half SPORT 'B' Multichannel 64-95 Select Register	23–58
Half SPORT 'A' Multichannel 96-127 Select Register	23–59
Half SPORT 'B' Multichannel 96-127 Select Register	23–60
Half SPORT 'A' Control 2 Register	23–61
Half SPORT 'B' Control 2 Register	23–62
Half SPORT 'A' Control Register	23–63
Half SPORT 'B' Control Register	23–71
Half SPORT 'A' Divisor Register	23–80
Half SPORT 'B' Divisor Register	23–81
Half SPORT 'A' Error Register	23–82
Half SPORT 'B' Error Register	23–84

Half SPORT 'A' Multichannel Control Register	23–86
Half SPORT 'B' Multichannel Control Register	23–88
Half SPORT 'A' Multichannel Status Register	23–90
Half SPORT 'B' Multichannel Status Register	23–91
Half SPORT 'A' Rx Buffer (Primary) Register	23–92
Half SPORT 'B' Rx Buffer (Primary) Register	23–93
Half SPORT 'A' Rx Buffer (Secondary) Register	23–94
Half SPORT 'B' Rx Buffer (Secondary) Register	23–95
Half SPORT 'A' Tx Buffer (Primary) Register	23–96
Half SPORT 'B' Tx Buffer (Primary) Register	23–97
Half SPORT 'A' Tx Buffer (Secondary) Register	23–98
Half SPORT 'B' Tx Buffer (Secondary) Register	23–99

Precision Clock Generators (PCG)

Features.....	24–1
Functional Description	24–1
ADSP-2156x PCG Register List.....	24–2
Internal Interface.....	24–2
Serial Clock	24–3
Frame Sync	24–3
Frame Sync Output.....	24–3
Divider Mode Selection.....	24–3
Phase Shift	24–4
Pulse Width	24–5
Default Pulse Width.....	24–5
Input Clock Source Considerations	24–5
Timing Example for I ² S Mode	24–6
Cross Mode Connections	24–6
Operating Modes.....	24–6
Normal Mode.....	24–6
Bypass Mode	24–7

One-Shot Mode.....	24-7
Audio System Example	24-8
Clock Configuration Examples	24-9
PCG Event Control	24-9
External Event Trigger	24-9
External Event Trigger Delay.....	24-10
Programming Model.....	24-10
Frame Sync Phase Setting	24-11
External Event Trigger	24-11
Debug Features	24-11
ADSP-2156x PCG Register Descriptions	24-11
Precision Clock A Control 0 Register	24-13
Precision Clock A Control 1 Register	24-14
Precision Clock B Control 0 Register	24-15
Precision Clock B Control 1 Register	24-16
Precision Clock C Control 0 Register	24-17
Precision Clock C Control 1 Register	24-18
Precision Clock D Control 0 Register	24-19
Precision Clock D Control 1 Register	24-20
Precision Clock Pulse Width Control 1 Register	24-21
Precision Clock Pulse Width Control 2 Register	24-23
Precision Clock Frame Sync Synchronization 1 Register	24-25
Precision Clock Frame Sync Synchronization 2 Register	24-28

Asynchronous Sample Rate Converter (ASRC)

Features.....	25-1
Functional Description	25-2
ADSP-2156x ASRC Register List	25-2
ASRC Interrupt List	25-2
ASRC Block Diagram.....	25-2

SRU Programming	25-3
Clocking	25-3
I/O Ports	25-3
De-Emphasis Filter	25-4
Mute Control	25-4
SRC Core	25-4
RAM FIFO	25-4
Digital Servo Loop	25-5
FIR Filter	25-5
Sample Rate Sensing	25-5
Digital Filter Group Delay	25-5
Data Format	25-6
Operating Modes	25-6
TDM Input Mode	25-6
TDM Output Mode	25-7
Matched-Phase Mode	25-7
Bypass Mode	25-8
De-Emphasis Mode	25-8
Dithering Mode	25-8
Muting Modes	25-9
Soft Mute	25-9
Hard Mute	25-9
Auto Mute	25-9
Interrupts	25-10
Sources	25-10
SRC Mute Out	25-10
Masking	25-10
Service	25-10
Programming Model	25-10
Debug Features	25-10

ADSP-2156x ASRC Register Descriptions	25–11
Control Register for ASRC 0 and 1	25–12
Control Register for ASRC 2 and 3	25–17
Mute Register	25–22
Ratio Register for ASRC 0 and 1	25–23
Ratio Register for ASRC 2 and 3	25–25
 Sony/Philips Digital Interface (S/PDIF)	
Features.....	26–1
ADSP-2156x SPDIF Register List	26–1
SRU Programming	26–3
S/PDIF Interrupt List.....	26–3
Clocking	26–3
S/PDIF Transmitter.....	26–3
Functional Description	26–3
Input Data Formats.....	26–5
Operating Modes.....	26–6
Full Serial Mode	26–6
Standalone Mode.....	26–6
Data Output Mode	26–7
S/PDIF Receiver	26–7
Functional Description	26–8
Clock Recovery.....	26–8
TDM Clock Output.....	26–9
Output Data Format	26–9
Channel Status	26–9
Operating Modes.....	26–9
Compressed or Non-linear Audio Data	26–10
Emphasized Audio Data.....	26–10
Single-Channel Double-Frequency Mode.....	26–10
Clock Recovery Modes	26–10

Interrupts.....	26–11
Sources	26–11
Transmit Block Start	26–11
Receiver Status	26–11
Receiver Error.....	26–11
Masking.....	26–11
Service	26–12
Programming Model.....	26–12
Programming the Transmitter	26–12
Programming the Receiver.....	26–12
Interrupted Data Streams on the Receiver	26–13
Debug Features	26–13
Loopback Routing	26–13
ADSP-2156x SPDIF Register Descriptions	26–13
Receive Control	26–15
Receive Status Register	26–17
Receive Status A0 Register	26–20
Receive Status B0 Register	26–21
Receive Status A1 Register	26–22
Receive Status B1 Register	26–23
Transmit Control Register	26–24
Transmit Status A0 Register	26–27
Transmit Status A1 Register	26–28
Transmit Status A2 Register	26–29
Transmit Status A3 Register	26–30
Transmit Status A4 Register	26–31
Transmit Status A5 Register	26–32
Transmit Status B0 Register	26–33
Transmit Status B1 Register	26–34
Transmit Status B2 Register	26–35

Transmit Status B3 Register	26–36
Transmit Status B4 Register	26–37
Transmit Status B5 Register	26–38
Transmit User Buffer A0 Register	26–39
Transmit User Buffer A1 Register	26–40
Transmit User Buffer A2 Register	26–41
Transmit User Buffer A3 Register	26–42
Transmit User Buffer A4 Register	26–43
Transmit User Buffer A5 Register	26–44
Transmit User Buffer B0 Register	26–45
Transmit User Buffer B1 Register	26–46
Transmit User Buffer B2 Register	26–47
Transmit User Buffer B3 Register	26–48
Transmit User Buffer B4 Register	26–49
Transmit User Buffer B5 Register	26–50
User Bit Update Register	26–51

Direct Memory Access (DMA)

DMA Channel Features	27–1
DMA Channel Functional Description	27–3
ADSP-2156x DMA Register List.....	27–3
ADSP-2156x DMA Channel List.....	27–4
DMA Definitions	27–5
Block Diagram.....	27–7
Architectural Concepts	27–9
DMA Channel SCB Interface.....	27–9
SCB Interface Signals	27–9
SCB Burst Transfers	27–10
Data Address Alignment	27–10
Descriptor Set Address Alignment	27–11
Peripheral Control Commands.....	27–11

Idle Command.....	27-12
Request-Data Command.....	27-12
Request-Data Urgent Command.....	27-12
Peripheral-Control Command Restrictions	27-12
DMA Channel Peripheral DMA Bus.....	27-12
Memory DMA and Triggering	27-13
Medium Band Width DMA Channel MMR Access Bus	27-15
DMA Channel Operation Flow.....	27-15
Startup Flow	27-15
Refresh Flow	27-17
Work Unit Transition Flow	27-18
Transfer Termination and Shutdown Flow	27-20
DMA Channel Errors.....	27-22
Status and Debug Errors	27-22
DMA Configuration Register Errors	27-23
Illegal Register Write During Run.....	27-23
Address Alignment Error.....	27-24
Memory Access Error	27-24
Trigger Overrun Error	27-24
Bandwidth-Monitor Error.....	27-24
Control Interface Error	27-24
DMA Operating Modes.....	27-24
Register-Based Flow Modes	27-25
Stop Mode.....	27-25
Autobuffer Mode.....	27-25
Descriptor-Based Flow Modes	27-25
Descriptor-Array Mode	27-26
Descriptor-List Mode	27-27
Descriptor-On-Demand Modes.....	27-27
Data Transfer Modes	27-28
Two-Dimensional DMA.....	27-28
DMA Channel Event Control.....	27-29

Event Signals	27–30
Work Unit State Events	27–31
Peripheral Interrupt Request Events	27–31
Peripheral Data Request Events	27–31
DMA Channel Triggers	27–32
Issuing Triggers	27–32
Waiting For Triggers	27–33
DMA Channel Programming Model	27–34
Mode Configuration	27–34
Register-Based Linear-Buffer Stop Flow Mode	27–34
Register-Based Autobuffer Flow Mode	27–35
Descriptor-Array Flow Mode	27–36
Descriptor-List Flow Mode	27–37
Register-Based Memory-to-Memory Transfer in Stop Flow Mode	27–38
Programming Concepts	27–40
Synchronization of Software and DMA	27–40
Interrupt and Trigger Event-Based Synchronization	27–40
Register Polling Based Synchronization	27–40
Descriptor Queues	27–41
Queues Using Event Generation for Every Descriptor Set	27–41
Queues Using Minimal Events	27–42
ADSP-2156x DMA Register Descriptions	27–43
Start Address of Current Buffer Register	27–45
Current Address Register	27–46
Bandwidth Limit Count Register	27–47
Bandwidth Limit Count Current Register	27–48
Bandwidth Monitor Count Register	27–49
Bandwidth Monitor Count Current Register	27–50
Configuration Register	27–51
Current Descriptor Pointer Register	27–59
Pointer to Next Initial Descriptor Register	27–60

Previous Initial Descriptor Pointer Register	27–61
Status Register	27–62
Inner Loop Count Start Value Register	27–65
Current Count (1D) or Intra-row XCNT (2D) Register	27–66
Inner Loop Address Increment Register	27–67
Outer Loop Count Start Value (2D only) Register	27–68
Current Row Count (2D only) Register	27–69
Outer Loop Address Increment (2D only) Register	27–70

Extended Memory DMA (EMDMA)

EMDMA Features	28–1
EMDMA Functional Description	28–1
ADSP-2156x EMDMA Register List.....	28–2
ADSP-2156x EMDMA Interrupt List	28–2
ADSP-2156x EMDMA Trigger List.....	28–3
DMA Addressing.....	28–3
DMA Burst Transfers	28–3
Transfer Control Block (TCB) Memory Storage.....	28–3
Chain Assignment	28–4
Starting Chain Loading	28–4
Buffered Chain Loading Register.....	28–4
TCB Storage.....	28–4
EMDMA Operating Modes.....	28–7
Standard DMA	28–8
Circular Buffered DMA.....	28–8
Chained DMA Mode.....	28–9
Data Direction On-the-Fly	28–9
Write Back Circular Index Pointer	28–10
Scatter/Gather DMA.....	28–10
Pre Modified Read/Write Index.....	28–12

Delay Line DMA.....	28–13
ADSP-2156x EMDMA Register Descriptions	28–15
External Base Address Register	28–17
Circular Buffer Length Register	28–18
Chain Pointer Register	28–19
Internal Count Register	28–20
External Count Register	28–21
External Memory DMA Control Register	28–22
Internal Index Register	28–26
External Index Register	28–27
Internal Modifier Register	28–28
External Modifier Register	28–29
Delay Line Tap Count Register	28–30
Tap List Pointer Register	28–31
Cyclic Redundancy Check (CRC)	
CRC Features.....	29–1
CRC Functional Description	29–2
ADSP-2156x CRC Register List.....	29–2
ADSP-2156x CRC Interrupt List	29–3
CRC Definitions	29–3
CRC Block Diagram.....	29–4
Peripheral DMA Bus	29–5
MMR Access Bus.....	29–5
Mirror Block.....	29–5
Data FIFO.....	29–5
DMA Request Generator.....	29–5
CRC Engine	29–5
Compare Logic	29–5
CRC Architectural Concepts.....	29–6

Look-up Table	29-6
Data Mirroring.....	29-6
FIFO Status and Data Requests.....	29-7
CRC Operating Modes	29-8
Data Transfer Modes	29-8
Memory Scan Compute-and-Compare Mode.....	29-9
Memory Scan Data Verify	29-10
Memory Transfer Compute-and-Compare Mode	29-10
Memory Transfer Data Fill Mode.....	29-10
CRC Event Control	29-11
Interrupt Signals.....	29-11
CRC Programming Model.....	29-11
CRC Mode Configuration	29-11
Look-up Table Generation	29-12
Core Driven Memory Scan Compute-and-Compare Mode	29-12
DMA Driven Memory Scan Compute-and-Compare Mode.....	29-14
Core Driven Memory Scan Data Verify Mode	29-15
DMA Driven Memory Scan Data Verify Mode	29-17
Core Driven Memory Transfer Compute-and-Compare Mode.....	29-18
DMA Driven Memory Transfer Compute-and-Compare Mode	29-20
DMA Driven Memory Transfer Data Fill Mode.....	29-21
ADSP-2156x CRC Register Descriptions	29-22
Data Compare Register	29-24
Control Register	29-25
Data Word Count Register	29-28
Data Count Capture Register	29-29
Data Word Count Reload Register	29-30
Data FIFO Register	29-31
Fill Value Register	29-32
Interrupt Enable Register	29-33
Interrupt Enable Clear Register	29-34

Interrupt Enable Set Register	29–35
Polynomial Register	29–36
CRC Current Result Register	29–37
CRC Final Result Register	29–38
Status Register	29–39

Housekeeping ADC (HADC)

HADC Features	30–1
HADC Functional Description	30–2
ADSP-2156x HADC Register List	30–2
ADSP-2156x HADC Interrupt List	30–2
ADSP-2156x HADC Trigger List.....	30–3
HADC Definitions	30–3
HADC Block Diagram	30–4
HADC Signal Descriptions	30–5
HADC Architectural Concepts.....	30–5
Converter Operation	30–5
Auto-Scan.....	30–6
Channel Sequence Programming.....	30–6
ADC Transfer Function.....	30–7
Results.....	30–7
HADC Operating Modes	30–7
HADC Event Control.....	30–7
HADC Programming Model	30–8
ADSP-2156x HADC Register Descriptions	30–8
Channel Mask Register	30–10
Control Register	30–11
Channel Data Registers	30–13
Interrupt Mask Register	30–14
Status Register	30–15

System Security

Security Features	31-1
Security Functional Description.....	31-2
Security Mode Configuration	31-4
Status and Error Signals.....	31-4

System Protection Unit (SPU)

SPU Features	32-1
SPU Functional Description	32-1
ADSP-2156x SPU Register List.....	32-1
ADSP-2156x SPU Interrupt List	32-2
Peripheral Register Write Protection.....	32-2
SPU Block Diagram.....	32-5
SPU Architectural Concepts	32-5
SPU Event Control.....	32-5
SPU Programming Model	32-6
SPU Mode Configuration.....	32-7
Locking Write-Protect Registers	32-7
Protecting a Peripheral	32-7
Configuring Security Privileges of a Peripheral	32-8
ADSP-2156x SPU Register Descriptions	32-8
Control Register	32-9
Secure Check Register	32-10
Secure Control Register	32-11
Secure Core Registers	32-13
Secure Peripheral Register	32-14
Status Register	32-15
Write Protect Register n	32-16
Write-Protect, Secure Peripheral, and Secure Core Registers	32-16
ADSP-2156x Specific Information.....	32-21

Security Packet Engine (PKTE)

PKTE Features.....	33-1
PKTE Functional Description	33-1
ADSP-2156x PKTE Register List.....	33-1
ADSP-2156x PKTE Interrupt List	33-3
PKTE Definitions	33-4
Cipher Module	33-5
Hash Module.....	33-5
Pseudo-Random Number Generator	33-6
Packet Engine Processing Details.....	33-8
Crypto Padding.....	33-8
Pad Generation and Insertion	33-9
Pad Types.....	33-9
Pad Length.....	33-10
Pad Verification and Consumption	33-12
Crypto and Hash Algorithms	33-14
IV Processing.....	33-17
ARC4 Processing.....	33-18
Hash State Loading	33-19
Sequence Number Processing.....	33-19
Sequence Number Processing in Extended SSL/TLS.....	33-19
Sequence Number Processing in DTLS.....	33-20
PKTE Block Diagram.....	33-23
PKTE Architectural Concepts	33-24
Packet Engine.....	33-24
Input/Output FIFO Buffers	33-24
Parallel Operations	33-24
DMA Controller	33-24
Interrupt Controller	33-25
Clock Controller	33-25
PKTE Operating Modes	33-25

Autonomous Ring Mode (ARM)	33–26
Target Command Mode (TCM)	33–27
Direct Host Mode (DHM)	33–27
PKTE Event Control	33–27
PKTE Interrupt Signals	33–27
PKTE Programming Model	33–30
PKTE Mode Configuration	33–33
PKTE Programming Concepts	33–33
Packet Engine Descriptor	33–33
Descriptor Processing	33–34
Descriptor Ownership	33–36
SA Record and State Record Structure	33–36
SA Record Structure	33–37
SA State Structure	33–39
ARC4 State Structure	33–39
Configuring Operations in the PKTE	33–40
Basic Operations and Decoding	33–40
Error Code Description	33–41
Extended Error Codes	33–42
Number Format	33–44
PKTE Programming Examples	33–45
Calculating SHA in Direct Host Mode	33–45
Performing AES Decryption in Direct Host Mode	33–46
ADSP-2156x PKTE Register Descriptions	33–47
Packet Engine ARC4 State Record Address	33–50
Starting Entry of 256-byte ARC4 State Buffer	33–51
Packet Engine Buffer Pointer Register	33–52
Packet Engine Buffer Threshold Register	33–53
Packet Engine Command Descriptor Ring Base Address	33–55
Packet Engine Command Descriptor Count Register	33–56
Packet Engine Command Descriptor Count Increment Register	33–57

Packet Engine Configuration Register	33–58
PE Clock Control Register	33–61
PKTE Continue Register	33–63
Packet Engine Control Register	33–64
Starting Entry of 256-byte Data Input/Output Buffer	33–69
Packet Engine Destination Address	33–70
Packet Engine DMA Configuration Register	33–71
Packet Engine Endian Configuration Register	33–73
Packet Engine Halt Control Register	33–75
Packet Engine Halt Status Register	33–77
Interrupt Mask Disable Register	33–80
Interrupt Mask Enable Register	33–82
Interrupt Masked Status Register	33–84
Packet Engine Input Buffer Count Register	33–86
Packet Engine Input Buffer Count Increment Register	33–87
Interrupt Configuration Register	33–88
Interrupt Clear Register	33–89
Interrupt Enable Register	33–91
Interrupt Unmasked Status Register	33–93
Packet Engine Length Register	33–95
Packet Engine Output Buffer Count Register	33–97
Packet Engine Output Buffer Count Decrement Register	33–98
Packet Engine Result Descriptor Ring Base Address	33–99
Packet Engine Result Descriptor Count Registers	33–100
Packet Engine Result Descriptor Count Decrement Registers	33–101
Packet Engine Ring Configuration	33–102
Packet Engine Ring Pointer Status	33–103
Packet Engine Ring Status	33–104
Packet Engine Ring Threshold Registers	33–105
Packet Engine SA Address	33–107

ARC4 i and j Pointer Register	33-108
SA Command 0	33-109
SA Command 1	33-114
SA Inner Hash Digest Registers	33-118
SA Key Registers	33-119
SA Initialization Vector Register	33-120
SA Outer Hash Digest Registers	33-121
SA Ready Indicator	33-122
SA Sequence Number Register	33-123
SA Sequence Number Mask Registers	33-124
SA SPI Register	33-125
Packet Engine Source Address	33-126
Packet Engine Status Register	33-127
Packet Engine State Record Address	33-131
State Hash Byte Count Registers	33-132
State Inner Digest Registers	33-133
State Initialization Vector Registers	33-134
Packet Engine User ID	33-135

Public Key Accelerator (PKA)

PKA Features	34-1
PKA Functional Description.....	34-1
ADSP-2156x PKA Register List	34-2
PKA Definitions	34-2
PKA Architectural Concepts.....	34-3
PKA Block Diagram	34-3
PKCP Vector Operations.....	34-4
Modular Exponentiation Operations	34-6
Modular Inversion	34-13
Modular Inversion with an Even Modulus.....	34-14

Modular Inversion with a Prime Modulus	34-14
ECC Operations	34-14
ADSP-2156x PKA Register Descriptions	34-20
PKA Vector_A Length	34-21
PKA Vector_A Address	34-22
PKA Vector_B Length	34-23
PKA Vector_B Address	34-24
PKA Compare Result	34-25
PKA Vector_C Address	34-26
PKA Most-Significant-Word of Divide Remainder	34-27
PKA Vector_D Address	34-28
PKA Function	34-29
Start of PKA RAM space	34-32
PKA Most-Significant-Word of Result Vector	34-33
PKA Bit Shift Value	34-34

Public Key Interrupt Controller (PKIC)

PKIC Functional Description	35-1
ADSP-2156x PKIC Register List.....	35-1
ADSP-2156x PKIC Interrupt List	35-2
PKIC Programming Model.....	35-2
PKIC Programming Concepts	35-2
ADSP-2156x PKIC Register Descriptions	35-3
Acknowledge Register	35-4
Enable Clear Register	35-5
Enable Control Register	35-6
Enable Set Register	35-7
Enabled Status Register	35-8
Polarity Control Register	35-9
Raw Status Register	35-10

Type Control Register	35-11
-----------------------------	-------

True Random Number Generator (TRNG)

TRNG Features	36-1
TRNG Functional Description	36-1
ADSP-2156x TRNG Register List.....	36-1
Random Number Generation	36-2
Locking Detection and Prevention.....	36-3
Run Testing.....	36-4
Monobit Testing.....	36-4
Poker Testing.....	36-5
Data for Tests	36-6
X9.31 Postprocessing.....	36-6
TRNG Block Diagram	36-6
TRNG Architectural Concepts	36-7
TRNG Operating Modes.....	36-8
TRNG Data Transfer Modes	36-9
TRNG Event Control.....	36-9
TRNG Interrupt Signals.....	36-9
ADSP-2156x TRNG Register Descriptions	36-10
TRNG Alarm Counter Register	36-12
TRNG Alarm Mask Register	36-14
TRNG Alarm Stop Register	36-15
TRNG Block Count Register	36-16
TRNG Configuration Register	36-17
Counter Register	36-19
TRNG Control Register	36-20
TRNG FRO De-tune Register	36-23
TRNG FRO Enable Register	36-24
TRNG Input Registers	36-25

TRNG Interrupt Acknowledge Register	36–26
Post-Process Key Registers	36–28
TRNG LFSR Access Register	36–29
TRNG LFSR Access Register	36–30
TRNG LFSR Access Register	36–31
TRNG Monobit Test Result Register	36–32
TRNG Output Registers	36–33
TRNG Poker Test Result Registers	36–34
TRNG Run Count Registers	36–35
TRNG Run Test State and Result Registers	36–36
TRNG Status Register	36–38
TRNG Test Register	36–40
TRNG Post-Process "V" Value Registers	36–43

Thermal Monitoring Unit (TMU)

TMU Features	37–1
TMU Functional Description	37–1
ADSP-2156x TMU Register List.....	37–1
ADSP-2156x TMU Interrupt List	37–2
ADSP-2156x TMU Trigger List	37–2
TMU Definitions	37–2
TMU Block Diagram	37–3
TMU Architectural Concepts	37–3
TMU and HADC	37–4
TMU Event Control.....	37–5
Status and Error Signals.....	37–5
TMU Programming Guidelines.....	37–5
Reading Temperature Data.....	37–6
ADSP-2156x TMU Register Descriptions	37–6
Alert High Limit Register	37–8

Alert Low Limit Register	37-9
Averaging Register	37-10
Temperature conversion blank register	37-11
TMU Control Register	37-12
Fault High Limit Register	37-13
Fault Low Limit Register	37-14
Gain Value Register	37-15
Interrupt Mask Register	37-16
Offset Register	37-17
Temperature Refresh Counter	37-18
Status Register	37-19
Temperature Value Register	37-20

FIR Accelerator (FIR)

Features.....	38-1
Clocking.....	38-1
Functional Description	38-2
ADSP-2156x FIR Register List.....	38-3
ADSP-2156x FIR Interrupt List	38-3
ADSP-2156x FIR Trigger List	38-4
Compute Block	38-4
Partial Sum Register	38-5
Delay Line Memory.....	38-5
Coefficient Memory	38-5
Prefetch Data Buffer.....	38-5
Processing Output.....	38-6
System Memory Storage	38-7
Coefficients and Input Buffer Storage	38-7
Single Rate Input Filtering	38-7
Decimation	38-7
Interpolation	38-8

Operating Modes	38-8
Single Rate Processing	38-8
Single Iteration.....	38-8
Floating-point Multi-Iteration.....	38-8
Window Processing	38-9
Multi-Rate Processing	38-9
Decimation.....	38-9
Interpolation	38-9
Floating-Point Data Format.....	38-10
Fixed-Point Data Format	38-10
Auto Configuration Mode (ACM)	38-10
Data Transfer	38-11
Chain Assignment	38-11
DMA Access	38-13
Burst Access Support	38-13
Accelerator TCB.....	38-13
Chain Pointer DMA.....	38-13
Programming Model.....	38-14
Legacy Mode	38-14
Auto Configuration Mode (ACM)	38-15
Debug Mode	38-17
Write to Local Memory	38-17
Read from Local Memory.....	38-17
Single-Step Mode	38-17
Computing FIR Output, Tap Length Greater than 4096.....	38-17
Debug Features	38-19
Local Memory Access	38-19
Single-Step Mode	38-19
Emulation Considerations	38-19
Interrupts.....	38-19

Sources	38–20
Window Complete	38–20
All Channels Complete	38–20
MAC Status	38–20
Service	38–21
ADSP-2156x FIR Register Descriptions	38–21
FIR Chain Pointer Register	38–23
FIR Coefficient Count Register	38–24
FIR Coefficient Index Register	38–25
FIR Coefficient Modifier Register	38–26
FIR Global Control Register	38–27
FIR Channel Control Register	38–30
Debug Address Register	38–32
FIR Debug Control Register	38–33
FIR Debug Data Read Register	38–34
FIR Debug Data Write Register	38–35
FIR DMA Status Register	38–36
FIR Input Data Base Register	38–38
FIR Input Data Count Register	38–39
FIR Input Data Index Register	38–40
FIR Input Data Modifier Register	38–41
FIR MAC Status Register	38–42
FIR Output Data Base Register	38–45
FIR Output Data Count Register	38–46
FIR Output Data Index Register	38–47
FIR Output Data Modifier Register	38–48
software control register 1	38–49
software control register 2	38–50
Secondary global control register	38–51

IIR Accelerator (IIR)

Features.....	39-1
Clocking	39-1
Functional Description	39-2
ADSP-2156x IIR Register List	39-3
ADSP-2156x IIR Interrupt List	39-4
ADSP-2156x IIR Trigger List.....	39-4
Multiply and Accumulate (MAC) Unit.....	39-4
Input Data and Biquad State	39-5
Coefficient Memory	39-5
Internal Memory Storage.....	39-5
Coefficient Memory Storage.....	39-5
Operating Modes.....	39-6
Window Processing Mode	39-6
40-Bit Floating-Point Mode	39-6
Save Biquad State Mode	39-6
Auto Configuration Mode (ACM)	39-7
Data Transfers.....	39-8
IIR Accelerator TCB.....	39-8
DMA Access.....	39-9
Burst Mode Access	39-10
Chain Pointer DMA	39-10
Interrupts.....	39-11
Sources	39-11
Window Complete.....	39-12
All Channels Complete	39-12
MAC Status	39-12
Service	39-12
Programming Model.....	39-13
ADSP-2156x IIR Register Descriptions	39-16
Chain Pointer Register	39-18

Coefficient Buffer Index Register	39–19
Coefficient Buffer Length Register	39–20
Coefficient Index Modifier Register	39–21
Global Control Register	39–22
Channel Control Register	39–25
IIR Debug Address Register	39–27
IIR Debug Control Register	39–28
IIR Debug Read Data High Register	39–29
IIR Debug Read Data Low Register	39–30
IIR Debug Write Data High Register	39–31
IIR Debug Write Data Low Register	39–32
DMA Status Register	39–33
Input Buffer Base Register	39–35
Input Data Index Register	39–36
Input Data Buffer Length Register	39–37
Input Data Index Modifier Register	39–38
MAC Status Register	39–39
Output Buffer Base Register	39–40
Output Data Buffer Index Register	39–41
IIR Output Data Buffer Length Register	39–42
IIR Output Data Index Modifier Register	39–43
Software Control Register1	39–44
Software Control Register2	39–45
Secondary Global Control Register	39–46

Boot ROM and Booting the Processor

SRAM Requirements	40–1
Preboot Operations.....	40–2
Start-up Sequence.....	40–3
Core Reset Sequencing	40–3

Core Start-up	40-3
Fault Configuration.....	40-4
L2 Controller Configuration	40-5
L2 Memory Initialization	40-5
Idle On Entry	40-6
SPU Configuration.....	40-6
SMPU Configuration	40-6
Secure Debug Key Processing	40-7
CGU Configuration	40-7
L1 Memory Initialization	40-9
Default Entry Point	40-10
<u>NO-BOOT Processing</u>	40-10
<u>SYS_RESOUT</u> Signal.....	40-10
DMC Configuration	40-10
Bypassing the Boot Process.....	40-12
Boot Mode Disable.....	40-12
Boot Command Customization	40-12
Boot Mode Specific SPU Configuration	40-13
Executing the Boot Mode	40-13
Boot Modes	40-14
No-Boot Mode	40-15
SPI Master Boot Mode	40-15
SPI Slave Boot Mode	40-20
Link Port Slave Boot Mode.....	40-24
UART Slave Boot Mode	40-26
OSPI Master Boot Mode.....	40-29
Boot Loader Stream	40-36
Block Types	40-40
Normal Block.....	40-40
First Block.....	40-40

Final Block	40-41
Indirect Block.....	40-41
Ignore Block.....	40-42
Fill Block.....	40-42
Init Block	40-43
Callback Block	40-44
Save Block	40-45
Single-Block Boot Streams	40-46
Direct Code Execution	40-46
Multi-Application Boot Streams	40-47
CRC32 Protection	40-49
Secure Boot.....	40-49
Terminology	40-51
Signing for Secure Boot Images	40-52
Secure Boot Image Types.....	40-52
Secure Boot Image Format.....	40-53
Secure Boot Image Attributes	40-56
Secure Boot Streams	40-56
Secure Debug Access.....	40-58
Boot ROM Errors and Failures	40-59
Boot ROM Programming Model	40-61
Boot Mode Driver API	40-61
Error Handler	40-63
Page Mode	40-64
Boot Hook Function	40-64
enum ROM_HOOK_CALL_CAUSE	40-64
Boot Return Feature	40-65
Boot Termination and Application Execution	40-65
Boot ROM OTP Customizations	40-66
API Reference	40-66

adi_rom_Boot()	40-66
adi_rom_BootKernel()	40-69
adi_rom_Crc32Init()	40-70
adi_rom_Crc32Poly()	40-70
adi_rom_GetAddress()	40-71
adi_rom_MemCompare()	40-71
adi_rom_MemCopy()	40-72
adi_rom_MemCrc()	40-73
adi_rom_MemDma()	40-73
adi_rom_MemFill()	40-75
adi_rom_PeriphDma()	40-76
adi_rom_otp_cfg()	40-77
adi_rom_otp_get()	40-77
adi_rom_otp_lock()	40-78
adi_rom_otp_pgm()	40-78
callback()	40-78
initcode()	40-81
adi_rom_idle_loop()	40-81
adi_rom_CguInit()	40-81
adi_rom_DmcPhyCalibration()	40-82
adi_rom_DmcInit()	40-82
adi_rom_ShaInit()	40-83
adi_rom_Sha()	40-83
Data Structures	40-86
struct ADI_ROM_BOOT_BUFFER	40-86
struct ADI_ROM_BOOT_CONFIG	40-86
struct ADI_ROM_BOOT_HEADER	40-96
struct ADI_ROM_BOOT_INTER_BUFFER	40-96
struct ADI_ROM_BOOT_INTER_BUFFERS	40-97
struct ADI_ROM_BOOT_LINKPORT	40-98

struct ADI_ROM_BOOT_MODES	40-98
struct ADI_ROM_BOOT_REGISTRY	40-99
struct ADI_ROM_BOOT_SPI	40-100
struct ADI_ROM_BOOT_OSPI	40-102
struct ADI_ROM_BOOT_UART	40-104
struct ADI_ROM_OTP_BOOT_CFG	40-105
struct ADI_ROM_OTP_BOOT_CGU_INFO	40-107
struct ADI_ROM_OTP_BOOT_CMD_INFO	40-110
struct ADI_ROM_OTP_BOOT_INFO	40-110
struct ADI_ROM_OTP_DMC_CONFIG.....	40-111
struct ROM_BOOT_DMA_INSTANCE.....	40-112
struct ROM_BOOT_MDMA	40-113
struct ROM_BOOT_MDMA_REGS	40-113
struct ROM_DMA_MDMA_CONFIG	40-114
struct ROM_DMA_PDMA_CONFIG	40-116
struct otp_data	40-117
Enumerations	40-120
enum ADI_ROM_BOOT_KEY_TYPE	40-120
enum ADI_ROM_BOOT_TYPE	40-120
enum OTPCMD.....	40-121
enum ROM_BOOT_MDMA_CRC_SUPPORT.....	40-123
enum ROM_CORE_ID	40-123
enum ROM_DMA_DONE_DETECT_METHOD.....	40-124
enum ROM_DMA_MDMA_ID.....	40-124
enum ROM_DMA_MDMA_OPERATION.....	40-125
enum ROM_DMA_PDMA_OPERATION	40-126
enum ROM_GETADDR_VALUE	40-126
enum ROM_HOOK_CALL_CAUSE	40-127
enum ROM_SB_IMAGE_TYPE.....	40-128

System Crossbars (SCB)

SCB Features	41-1
SCB Functional Description	41-1
ADSP-2156x SCB0 Register List.....	41-2
ADSP-2156x SCB1 Register List.....	41-5
ADSP-2156x SCB3 Register List.....	41-5
SCB Architectural Concepts	41-5
SCB Block Diagram	41-6
SCB Block Diagram	41-7
System Crossbars	41-13
SCB Bus Master IDs	41-14
SCB Programming Model.....	41-16
SCB Programming Concepts.....	41-18
QoS Programming	41-21
ADSP-2156x SCB0 Register Descriptions	41-22
Accelerator Channel 0 Read Quality of Service Register	41-26
Accelerator Channel 0 Write Quality of Service Register	41-27
Accelerator Channel 1 Read Quality of Service Register	41-28
Accelerator Channel 1 Write Quality of Service Register	41-29
CRC0 Channel 0 Read Quality of Service Register	41-30
CRC0 Channel 0 Write Quality of Service Register	41-31
CRC0 Channel 1 Read Quality of Service Register	41-32
CRC0 Channel 1 Write Quality of Service Register	41-33
CRC1 Channel 0 Read Quality of Service Register	41-34
CRC1 Channel 0 Write Quality of Service Register	41-35
CRC1 Channel 1 Read Quality of Service Register	41-36
CRC1 Channel 1 Write Quality of Service Register	41-37
Crypto Read Quality of Service Register	41-38
CRYPTO Write Quality of Service Register	41-39
DBG Read Quality of Service Register	41-40

DBG Write Quality of Service Register	41-41
DLDMA0 Channel 0 Read Quality of Service Register	41-42
DLDMA0 Channel 0 Write Quality of Service Register	41-43
DLDMA0 Channel 1 Read Quality of Service Register	41-44
DLDMA0 Channel 1 Write Quality of Service Register	41-45
DLDMA1 Channel 0 Read Quality of Service Register	41-46
DLDMA1 Channel 0 Write Quality of Service Register	41-47
DLDMA1 Channel 1 Read Quality of Service Register	41-48
DLDMA1 Channel 1 Write Quality of Service Register	41-49
ETR Read Quality of Service Register	41-50
ETR Write Quality of Service Register	41-51
HSMDMA Channel 0 Read Quality of Service Register	41-52
HSMDMA Channel 0 Write Quality of Service Register	41-53
HSMDMA Channel 1 Read Quality of Service Register	41-54
HSMDMA Channel 1 Write Quality of Service Register	41-55
LP0 Read Quality of Service Register	41-56
LP0 Write Quality of Service Register	41-57
LP1 Read Quality of Service Register	41-58
LP1 Write Quality of Service Register	41-59
MLB Read Quality of Service Register	41-60
MLB Write Quality of Service Register	41-61
MSMDMA Channel 0 Read Quality of Service Register	41-62
MSMDMA Channel 0 Write Quality of Service Register	41-63
MSMDMA Channel 1 Read Quality of Service Register	41-64
MSMDMA Channel 1 Write Quality of Service Register	41-65
SH0 DPORT Read Quality of Service Register	41-66
SH0 DPORT Write Quality of Service Register	41-67
SH0 IPORT Read Quality of Service Register	41-68
SH0 IPORT Write Quality of Service Register	41-69
SH0 MMR Read Quality of Service Register	41-70

SH0 MMR Write Quality of Service Register	41-71
SP0A Read Quality of Service Register	41-72
SP0A Write Quality of Service Register	41-73
SP0B Read Quality of Service Register	41-74
SP0B Write Quality of Service Register	41-75
SP1A Read Quality of Service Register	41-76
SP1A Write Quality of Service Register	41-77
SP1B Read Quality of Service Register	41-78
SP1B Write Quality of Service Register	41-79
SP2A Read Quality of Service Register	41-80
SP2A Write Quality of Service Register	41-81
SP2B Read Quality of Service Register	41-82
SP2B Write Quality of Service Register	41-83
SP3A Read Quality of Service Register	41-84
SP3A Write Quality of Service Register	41-85
SP3B Read Quality of Service Register	41-86
SP3B Write Quality of Service Register	41-87
SP4A Read Quality of Service Register	41-88
SP4A Write Quality of Service Register	41-89
SP4B Read Quality of Service Register	41-90
SP4B Write Quality of Service Register	41-91
SP5A Read Quality of Service Register	41-92
SP5A Write Quality of Service Register	41-93
SP5B Read Quality of Service Register	41-94
SP5B Write Quality of Service Register	41-95
SP6A Read Quality of Service Register	41-96
SP6A Write Quality of Service Register	41-97
SP6B Read Quality of Service Register	41-98
SP6B Write Quality of Service Register	41-99
SP7A Read Quality of Service Register	41-100

SP7A Write Quality of Service Register	41-101
SP7B Read Quality of Service Register	41-102
SP7B Write Quality of Service Register	41-103
SPI0 RX Read Quality of Service Register	41-104
SPI0 RX Write Quality of Service Register	41-105
SPI0 TX Read Quality of Service Register	41-106
SPI0 TX Write Quality of Service Register	41-107
SPI1 RX Read Quality of Service Register	41-108
SPI1 RX Write Quality of Service Register	41-109
SPI1 TX Read Quality of Service Register	41-110
SPI1 TX Write Quality of Service Register	41-111
SPI2 RX Read Quality of Service Register	41-112
SPI2 RX Write Quality of Service Register	41-113
SPI2 TX Read Quality of Service Register	41-114
SPI2 TX Write Quality of Service Register	41-115
UART0 RX Read Quality of Service Register	41-116
UART0 RX Write Quality of Service Register	41-117
UART0 TX Read Quality of Service Register	41-118
UART0 TX Write Quality of Service Register	41-119
UART1 TX Read Quality of Service Register	41-120
UART1 RX Write Quality of Service Register	41-121
UART1 TX Read Quality of Service Register	41-122
UART1 TX Write Quality of Service Register	41-123
UART2 RX Read Quality of Service Register	41-124
UART2 RX Write Quality of Service Register	41-125
UART2 TX Read Quality of Service Register	41-126
UART2 TX Write Quality of Service Register	41-127
ADSP-2156x SCB1 Register Descriptions	41-127
Master Synchronization Mode Register	41-128
ADSP-2156x SCB3 Register Descriptions	41-128

APB DCLK 0 Synchronization Mode Register	41-129
ADSP-2156x SCB4 Register Descriptions	41-129
Comp Id 0	41-131
Comp Id 1	41-132
Comp Id 2	41-133
Comp Id 3	41-134
Fabric S2port Ib.fn Mod	41-135
Fabric S2port Ib.read Qos	41-136
Fabric S2port Ib.write Qos	41-137
Fir Ch0 Ib.fn Mod	41-138
Fir Ch0 Ib.fn Mod2	41-139
Fir Ch0 Ib.read Qos	41-140
Fir Ch0 Ib.write Qos	41-141
Fir Ch1 Ib.fn Mod	41-142
Fir Ch1 Ib.fn Mod2	41-143
Fir Ch1 Ib.read Qos	41-144
Fir Ch1 Ib.write Qos	41-145
Iir Ch0 Ib.fn Mod	41-146
Iir Ch0 Ib.fn Mod2	41-147
Iir Ch0 Ib.read Qos	41-148
Iir Ch0 Ib.write Qos	41-149
Iir Ch1 Ib.fn Mod	41-150
Iir Ch1 Ib.fn Mod2	41-151
Iir Ch1 Ib.read Qos	41-152
Iir Ch1 Ib.write Qos	41-153
Periph Id 0	41-154
Periph Id 1	41-155
Periph Id 2	41-156
Periph Id 3	41-157
Periph Id 4	41-158

Periph Id 5	41-159
Periph Id 6	41-160
Periph Id 7	41-161
Sharc Dport.fn Mod	41-162
Sharc Dport.read Qos	41-163
Sharc Dport.write Qos	41-164
ADSP-2156x SCB5 Register Descriptions	41-164
Comp Id 0	41-166
Comp Id 1	41-167
Comp Id 2	41-168
Comp Id 3	41-169
Periph Id 0	41-170
Periph Id 1	41-171
Periph Id 2	41-172
Periph Id 3	41-173
Periph Id 4	41-174
Periph Id 5	41-175
Periph Id 6	41-176
Periph Id 7	41-177
.....	41-178
Spi2 S Ib.fn Mod	41-179
Spi2 S Ib.fn Mod Iss Bm	41-180
Spif M.fn Mod	41-181

System Watchpoint Unit (SWU)

SWU Features.....	42-1
SWU Functional Description.....	42-1
ADSP-2156x SWU Register List	42-1
SWU Interrupt List	42-2
SWU Trigger List	42-2

SWU Definitions.....	42-2
SWU Architectural Concepts.....	42-2
SWU-to-SCB Interface.....	42-2
SWU Block Diagram.....	42-2
SCB Interface Block	42-3
MMR Interface Block	42-3
SWU Operating Modes	42-3
Bandwidth Mode.....	42-3
Watchpoint Mode.....	42-3
Match Block	42-3
Scaling.....	42-4
SWU Event Control	42-4
SWU Interrupts.....	42-4
SWU Status and Errors.....	42-5
Triggers	42-5
SWU Programming Model	42-5
SWU Mode Configuration	42-6
Configuring the SWU for Bandwidth Mode	42-6
Configuring the SWU for Watchpoint Mode	42-7
ADSP-2156x SWU Register Descriptions	42-8
Count Register n	42-9
Control Register n	42-10
Current Register n	42-15
Global Control Register	42-16
Global Status Register	42-17
Bandwidth History Register n	42-21
ID Register n	42-22
Lower Address Register n	42-23
Target Register n	42-24
Upper Address Register n	42-25

Memory Error Protection Unit (MEPU)

Memory Error Controller (MEC)	43-1
MEC Features	43-1
Parity Controller (PCTL)	43-2
PCTL Features.....	43-2
MEC Functional Description.....	43-2
ADSP-2156x MEC Register List	43-2
ADSP-2156x MEC Interrupt List	43-3
ADSP-2156x MEC Trigger List.....	43-4
MEPU Block Diagram	43-4
MEC Block Diagram	43-4
PCTL Block Diagram.....	43-5
MEC Architectural Concepts.....	43-5
MEC Configuration	43-5
PCTL Integration	43-7
ADSP-2156x MEC Register Descriptions	43-7
Component ID0 Register	43-9
Component ID1 Register	43-10
Component ID2 Register	43-11
Component ID3 Register	43-12
Clear Register	43-13
ECC Error Control Register	43-14
ECC Error Interrupt Mask Register	43-15
ECC Error Status Register	43-16
ECC Error Interrupt Request Global Control Register	43-17
ECC Error Interrupt Request Global Status Register	43-18
Parity Error Interrupt Request Global Control Register	43-19
Parity Error Interrupt Request Global Status Register	43-20
Parity Error Control Register	43-21
Parity Error Interrupt Mask Register	43-23

Parity Error Status Register	43–25
Peripheral ID0 Register	43–27
Peripheral ID1 Register	43–28
Peripheral ID2 Register	43–29
Peripheral ID3 Register	43–30
Peripheral ID4 Register	43–31
Peripheral ID5 Register	43–32
Peripheral ID6 Register	43–33
Peripheral ID7 Register	43–34

System Debug and Trace Unit (DBG)

DBG Features	44–1
DBG Functional Description.....	44–2
ADSP-2156x CSPFT Register List.....	44–2
ADSP-2156x TAPC Register List.....	44–3
DBG Block Diagram	44–3
DBG Definitions.....	44–4
Test Access Port Controller (TAPC)	44–5
Debug Access Ports.....	44–5
Trace Unit	44–6
Programmable Flow Trace (CSPFT).....	44–6
System Trace Module (STM)	44–7
Embedded Cross Trigger (ECT)	44–7
CTI Debug Trigger Tables.....	44–8
ADSP-2156x CSPFT Register Descriptions	44–10
Address Comparator Access Type Register	44–13
Address Comparator Value Register	44–14
Authentication Status Register	44–15
Configuration Code Extension Register	44–16
Component ID0 Register	44–17

Component ID1 Register	44-18
Component ID2 Register	44-19
Component ID3 Register	44-20
Context ID Comparator Mask Register	44-21
Context ID Comparator Value	44-22
Claim Tag Clear Register	44-23
Claim Tag Set Register	44-24
Counter Enable Event Register	44-25
Counter Reload Event Register	44-28
Counter Reload Value Register	44-31
Counter Value Register	44-32
Main Control Register	44-33
Device Type Identifier Register	44-35
External Output Event Register	44-36
Hardware Feature Register	44-39
Lock Access Register	44-41
Lock Status Register	44-42
Peripheral ID0 Register	44-43
Peripheral ID1 Register	44-44
Peripheral ID2 Register	44-45
Peripheral ID3 Register	44-46
Peripheral ID4 Register	44-47
Status Register	44-48
Synchronization Frequency Register	44-49
TraceEnable Control Register	44-50
TraceEnable Event Register	44-51
CoreSight Trace ID Register	44-54
Trigger Event Register	44-55
TraceEnable Start/Stop Control Register	44-58
ADSP-2156x TAPC Register Descriptions	44-59

Debug Control Register	44-61
IDCODE Register	44-63
Secure Debug Key 0 Register	44-64
Secure Debug Key 1 Register	44-65
Secure Debug Key 2 Register	44-66
Secure Debug Key 3 Register	44-67
Secure Debug Key Control Register	44-68
Secure Debug Key Status Register	44-69
USERCODE Register	44-70

ADSP-2156x Register List

Preface

Thank you for purchasing and developing systems using an ADSP-2156x SHARC+[®] processor from Analog Devices, Inc.

Purpose of This Manual

The *ADSP-2156x SHARC+ Processor Hardware Reference* provides architectural information about the ADSP-2156x processors. This hardware reference provides the main architectural information about these processors. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes that they support.

For timing, electrical, and package specifications, refer to the *SHARC+ Single Core High Performance DSP ADSP-21563/21565/21567/21569 Data Sheet*.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. The manual assumes the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts, such as programming reference books and data sheets, that describe their target architecture.

What's New in This Manual

This manual is revision 1.0 of the *ADSP-2156x SHARC+ Processor Hardware Reference*. The following changes are included in this edition:

- Added Introduction, HADC, and TMU chapters
- Updated the Oscillator Watchdog section of the CGU chapter
- Updated the Programming the DMC Controller section of the DMC chapter
- Updated the OTP Layout section of the OTPC chapter
- Updated the Internal Timer Connections information in the GP Timer chapter
- Updated the Boot ROM and Booting the Processor chapter
- Updated Digital Filter Group Delay section of the ASRC chapter

Technical or Customer Support

You can reach customer and technical support for processors from Analog Devices in the following ways:

- Post your questions in the processors and DSP support community at *EngineerZone*[®]:

<http://ez.analog.com/community/dsp>

- Submit your questions to technical support at *Connect with ADI Specialists*:

<http://www.analog.com/support>

- E-mail your questions about software/hardware development tools to:

processor.tools.support@analog.com

- E-mail your questions about processors and DSPs to:

processor.support@analog.com (world wide support)

processor.china@analog.com (China support)

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

<http://www.analog.com/adi-sales>

- Send questions by mail to:

Analog Devices, Inc.

Three Technology Way

P.O. Box 9106

Norwood, MA 02062-9106 USA

Product Information

Product information can be obtained from the Analog Devices Web site and CrossCore Embedded Studio[®] (CCES) online Help system.

Analog Devices Web Site

The Analog Devices Web site, <http://www.analog.com>, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to: http://www.analog.com/processors/technical_library The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit <http://ez.analog.com> to sign up.

Supported Processors

The following is the list of Analog Devices, Inc. processors supported by the CCES development tools suite.

Blackfin[®] (ADSP-BF7xx) Processors

The name *Blackfin+* refers to the enhanced fixed-point Blackfin core architecture featured by the ADSP-BF70x processor product line, which is a family of 16-bit embedded processors.

Blackfin[®] (ADSP-BF6xx/BF5xx) Processors

The name *Blackfin* refers to the fixed-point core architecture featured on the following processors: ADSP-BF5xx and ADSP-BF6xx.

SHARC[®] (ADSP-21xxx) Processors

The name *SHARC* refers to the high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-2116x, ADSP-2126x, ADSP-213xx, and ADSP-214xx. These processors can be used in speech, sound, graphics, and imaging applications.

SHARC+[®] (ADSP-SC5xx, ADSP-215xx) Processors

The name *SHARC+* refers to the enhanced high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-215xx/ADSP-SC5xx. The connected SHARC+ ADSP-SC5xx processors also contain an Arm[®] Cortex[®]-A5 core. These products can be used in speech, sound, graphics, and imaging applications.

The following is the list of Analog Devices, Inc. processors supported by the IAR Embedded WorkBench[®] development tools. For information about the IAR Embedded WorkBench product and software download, go to <http://www.iar.com/en/Products/IAR-Embedded-Workbench> .

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
<i>File > Close</i>	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the <i>Close</i> command appears on the <i>File</i> menu).
{this that}	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as <i>this</i> or <i>that</i> . One or the other is required.
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional <i>this</i> or <i>that</i> .
[this, ...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, and feature names are in text with Letter Gothic font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
NOTE:	<i>NOTE:</i> For correct operation, ... A note provides supplementary information on a related topic. In the online version of this book, the word <i>NOTE:</i> appears instead of this symbol.
CAUTION:	<i>CAUTION:</i> Incorrect device operation may result if ... <i>CAUTION:</i> Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word <i>CAUTION:</i> appears instead of this symbol.
ATTENTION:	<i>ATTENTION:</i> Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word <i>ATTENTION:</i> appears instead of this symbol.
Registers/Bits	All registers and bits in this manual are linked (clickable) to their respective descriptions in the "Register Descriptions" of each chapter.
Miscellaneous Conventions	Interrupt and internal signals are shown in all caps with no other formatting. For example the SPDIFn_RX or SCLK signal or the PKTE0_IRQ interrupt. An overbar denotes an active-low signal as in $\overline{\text{SYS_FAULT}}$.

Register Documentation Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top with the short form of the name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name.
- The reset value appears in binary in the individual bits and in hexadecimal to the left of the register.
- Bits marked *X* have an unknown reset value. Consequently, the reset value of registers that contain such bits is undefined or dependent on pin values at reset.
- Shaded bits are reserved

NOTE: To ensure upward compatibility with future implementations, write back the value that is read for reserved bits in a register, unless otherwise specified.

Register description tables use the following conventions:

- Each bit's or bit field's access type appears beneath the bit number in the table in the form (read-access/write-access). The access types include:
 - R = read, RC = read clear, RS = read set, R0 = read zero, R1 = read one, Rx = read undefined
 - W = write, NW = no write, W1C = write one to clear, W1S = write one to set, W0C = write zero to clear, W0S = write zero to set, WS = write to set, WC = write to clear, W1A = write one action
- Many bit and bit field descriptions include enumerations, identifying bit values and related functionality. Unless otherwise indicated (with a prefix), these enumerations are decimal values.

1 Introduction

The ADSP-2156x processors are members of the SHARC+ family of products.

As shown in the *ADSP-2156x Functional Block Diagram*, by integrating a rich set of industry-leading system peripherals and memory (see product data sheet), the SHARC+ processor is the platform of choice for next-generation applications that require RISC-like programmability, multimedia support, and leading-edge signal processing in one integrated package. These applications span a wide array of markets, from automotive and pro-audio to industrial-based applications that require high floating-point performance.

NOTE: For specific product configurations, refer to the *SHARC+ Single Core High Performance DSP ADSP-21562/ADSP-21563/21565/21567/21569* data sheet.

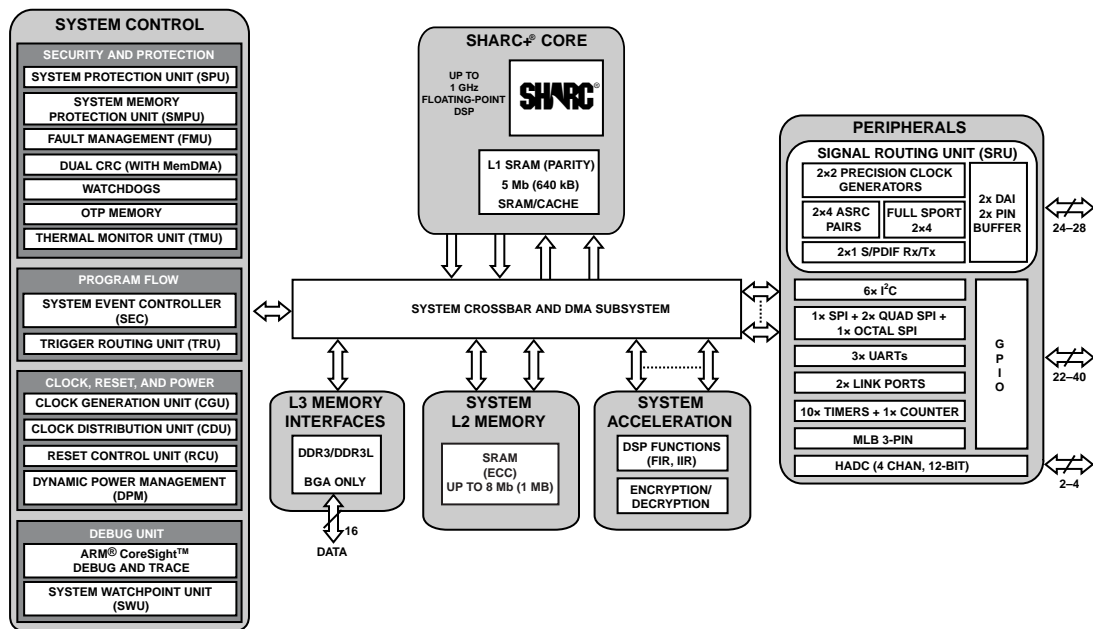


Figure 1-1: ADSP-2156x Functional Block Diagram

Power and Clock Management (DPM/RCU/CGU/CDU)

The processor contains four modules that control power and clocking.

Dynamic Power Management (DPM)

The [Dynamic Power Management \(DPM\)](#) unit of the processor controls transitions between different power-saving modes.

Reset Control Unit (RCU)

The [Reset Control Unit \(RCU\)](#) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. The RCU supports separate reset control for various chip sub-systems. For deterministic operation programmers should ensure there is no activity between separate chip sub-systems during reset activity. More global reset options are also supported. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. Programs must ensure that there is no pending system activity involving the core prior to reset.

Clock Generation Unit (CGU)

The [Clock Generation Unit \(CGU\)](#) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock that runs at a frequency that is a multiple of the SYS_CLKINx input clock frequency. It also generates all on-chip clocks and synchronization signals. The PCU allows the application software to control the PLL module operation. All clocks derived from the CGU are then forwarded to the CDU for distribution.

Clock Distribution Unit (CDU)

The [Clock Distribution Unit \(CDU\)](#) consists of an array of software-configurable multiplexors that select clocks originated from up to four different clock sources that are generated from the CGUs. Unused input clocks are grounded internally and never selected. The output clock signal for each multiplexor is assigned to one or more destinations within the processor.

System Interrupts and Triggers (SEC/TRU)

The SEC and TRU modules provide interrupt and trigger management functions for the processor.

System Event Controller (SEC)

The SEC manages the configuration of all system event sources and the propagation of system events to the processor core and the system fault interface. The SEC arbitrates among all pending system interrupt requests and presents the highest priority interrupt to the core for processing.

Trigger Routing Unit (TRU)

The [Trigger Routing Unit \(TRU\)](#) provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes

- Software triggering
- Synchronization of concurrent activities

System Memory (L2CTL/DMC/OTPC/SMPU)

The following sections describe the memory architecture of the processor. For more information, refer to the processor data sheet.

L2 Memory Controller (L2CTL)

The [L2 System Memory](#) has significant bandwidth for core accesses. However, L2 responds slower to core accesses than L1 memories. Due to sophisticated data integrity protection and write protection, L2 SRAM is also ideal for data and instructions critical for safe operation of the application.

Dynamic Memory Controller (DMC)

The [Dynamic Memory Controller \(DMC\)](#) provides a glueless interface between DDR3/DDR3L SDRAMs and the system crossbar interface (SCB). The DMC enables execution of instructions from, as well as transfer of data to and from DDR3, DDR3L SDRAM, respectively.

The DMC is partitioned in a manner that allows reconfiguration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the protocol controller. An access and operation reordering mechanism is incorporated as an efficiency controller. An SCB slave interface is provided to interface with the on-chip interconnect. This interface results in an efficient slave implementation owing to its out-of-order transaction capabilities.

The DMC supports access to the external memory by core and DMA accesses.

One-Time Programmable Memory Controller (OTPC)

The [One-Time Programmable Memory Controller \(OTPC\)](#) module is a complete system integrating an OTP memory core with a programming controller, charge pump, and voltage regulator. A built-in Hamming Code Error Correction (ECC), and a fully implemented double-redundant program or read scheme protect the OTP data.

System Memory Protection Unit (SMPU)

The [System Memory Protection Unit \(SMPU\)](#) provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

On the ADSP-2156x, four SMPU instances are available to protect the L2, external memory (DMC), and SPI Flash address space.

Direct Memory Access (DMA/MDMA/EMDMA/CRC)

DMA Controller (DMA)

The processors use [Direct Memory Access \(DMA\)](#) to transfer data within memory spaces or between a memory space and a peripheral. The processors can specify data transfer operations and return to normal processing while the fully integrated DMA controller carries out the data transfers independent of processor activity.

DMA transfers can occur between memory and a peripheral or between one memory and another memory. Each Memory-to-memory DMA stream uses two channels, where one channel is the source channel, and the second is the destination channel. Most peripherals have at least one dedicated DMA channel associated with them.

All DMAs can transport data to and from all on-chip and off-chip memories. Programs can use two types of DMA transfers, descriptor-based or register-based. Register-based DMA allows the processors to directly program DMA control registers to initiate a DMA transfer. On completion, the control registers may be automatically updated with their original setup values for continuous transfer. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based DMA transfers allow multiple DMA sequences to be chained together and a DMA channel can be programmed to automatically set up and start another DMA transfer after the current sequence completes.

Memory DMA Controllers (MDMA)

The processor supports a variety of [Memory DMA and Triggering](#) memory-to-memory DMA operations which include:

- Two enhanced memory DMA channels with CRC protection (32-bit bus width, run on SYSCLK)
- One enhanced memory DMA channel (32-bit bus width, runs on SYSCLK)
- Two memory DMA channels (64-bit bus width, run on SYSCLK)
- Two EMDMA channels (32-bit bus width, run on SYSCLK)

Extended Memory DMA (EMDMA)

The [Extended Memory DMA \(EMDMA\)](#) engine can be used in applications that copy data in a non-sequential manner. This includes delay lines, scatter and gather, and circular access types.

Unlike previous SHARC processors which contained external port DMA, the current EMDMA module can access all memory locations (L1/L2/L3) for source and destination DMA operations.

Cyclic Redundancy Check (CRC)

The [Cyclic Redundancy Check \(CRC\)](#) peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error. The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by core write transactions.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the look-up table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

Peripherals

The SHARC+ processor contains a rich set of industry leading system peripherals. The processor is the platform of choice for applications that require RISC-like programmability, multimedia support, and leading edge signal processing in one integrated package. These applications span a wide array of markets, including automotive, pro audio, and industrial-based applications that require high floating-point performance. These peripherals are described in the following sections.

- [General-Purpose I/O \(GPIO\) Peripherals](#)
- [DAI/SRU Peripherals](#)

General-Purpose I/O (GPIO) Peripherals

The SHARC+ processors feature up to 40 general-purpose I/O pins mapped across up to three ports (PORT A through PORT C). Each pin can be configured individually to serve as a GPIO pin or as a peripheral-specific pin.

GPIO Ports (PORT)

When configured in the default GPIO mode, the PORT pins allow for the processor to interface to system components to provide handshaking functionality as either inputs or outputs. When in output mode, open-drain output is supported. A single MMR access can be used to sense or set individual pins or a complete port of 16 pins.

Additionally, each GPIO pin can optionally be configured to raise a system interrupt on the processor via a dedicated pin interrupt (PINT) block, and all peripheral functions are controlled via a set of port multiplexing registers, with specific settings defined in the processor data sheet.

Link Ports (LP)

The [Link Port \(LP\)](#) allow the processor to connect to other processors or peripheral link ports using a simple communication protocol for high-speed parallel data transfer. This peripheral allows various I/O peripheral interconnection schemes to I/O peripheral devices, as well as co-processing and multiprocessing schemes.

Serial Peripheral Interface Ports (SPI)

The [Serial Peripheral Interface \(SPI\)](#) is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

Octal Serial Peripheral Interface (OSPI)

The [Octal Serial Peripheral Interface \(OSPI\)](#) controller provides access to serial flash devices. It is designed to access flash devices with its modes enabling efficient communication, with minimum overhead on software. Standard SPI is supported along with high performance dual, quad, and octal SPI modes where data can be transferred on up to 8 data pins. Along with normal mode of operation in single transfer (STR) mode, the OSPI controller supports dual transfer rate (DTR) mode, where data, address, and commands are transferred on both edges of the serial clock. With direct access mode support, any read/write to the OSPI memory space triggers a read/write to flash memory.

Universal Asynchronous Receiver/Transmitter (UART)

The [Universal Asynchronous Receiver/Transmitter \(UART\)](#) module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. The UART includes interrupt-handling hardware. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

General-Purpose Counter (CNT)

The [General-Purpose Counter \(CNT\)](#) converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

Media Local Bus 3-Pin (MLB)

The [Media Local Bus \(MLB\)](#)[®] is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The MLB topology

supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

DAI/SRU Peripherals

The [Digital Audio Interface \(DAI\)](#) (DAIn) are comprised of groups of identical peripherals and their respective Signal Routing Units (SRUn). The SRU connects inputs and outputs of the DAI peripherals with each other and to the external pins. This configuration allows peripherals to be interconnected to accommodate a wide variety of systems without making external pin connections.

The [DAI Routing Capabilities](#) section provides an overview of the different routing capabilities for the DAI unit. An example for the eight SPORTs is shown in the following figure.

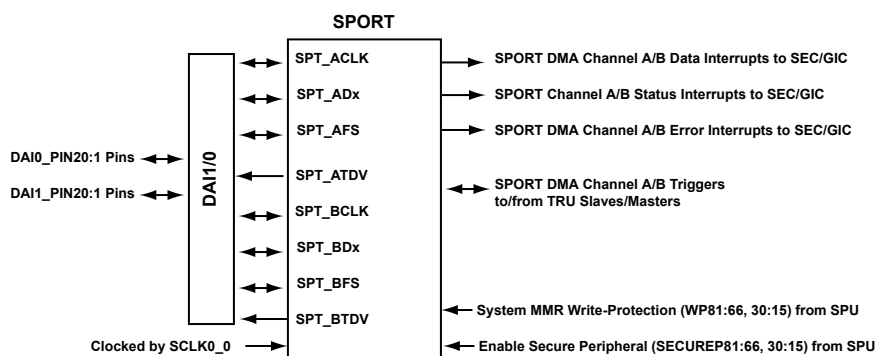


Figure 1-2: SPORT Block Diagram

The following additional peripherals are connected using the DAI/SRU.

- Eight [Asynchronous Sample Rate Converter \(ASRC\)](#) blocks
- Two [Sony/Philips Digital Interface \(S/PDIF\)](#) transmit/receive blocks
- Four [Precision Clock Generators \(PCG\)](#)

Two-Wire Interface (TWI)

The processor has a [Two-Wire Interface \(TWI\)](#), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

System Accelerators (FIR/IIR)

The following sections provide information about the high-performance acceleration engines on the processor.

FIR Accelerator (FIR)

[FIR Accelerator \(FIR\)](#) filters are frequently used in DSP applications. The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core. FIR filters are used in a wide array of applications including multi-rate processing with an interpolator or decimator.

IIR Accelerator (IIR)

The processor includes an [IIR Accelerator \(IIR\)](#) implemented in hardware that reduces the processing load on the core, freeing it up for other tasks.

Security and Protection (SPU/PKTE/PKIC/PKA/TRNG)

The following modules provide system safety and security.

System Protection Unit (SPU)

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The [System Protection Unit \(SPU\)](#) restricts access to certain MMRs, similar to the functionality of a semaphore.

The SPU also protects peripherals based on security settings. It is part of the overall security infrastructure of the processor.

Security Packet Engine (PKTE)

The [Security Packet Engine \(PKTE\)](#) is a security packet engine designed to off-load the host processor to improve the speed of applications requiring cryptographic processing. The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation.

Public Key Accelerator (PKA)

The [Public Key Accelerator \(PKA\)](#) helps offload computationally-intensive operations commonly found in public key cryptography algorithms. The PKA also contains hardware logic to automatically zero out the PKA RAM buffer to clear out any information that is considered sensitive or secure.

Public Key Interrupt Controller (PKIC)

The [Public Key Interrupt Controller \(PKIC\)](#) is a common interrupt controller shared with the True Random Number Generator. The host processor configures the PKIC to generate interrupts when certain operations are complete or interrupts are caused by errors.

True Random Number Generator (TRNG)

The [True Random Number Generator \(TRNG\)](#) engine provides a true, non-deterministic, noise source for generating keys, Initialization Vectors (IVs), and other random number requirements. Other non-cryptographic purposes include statistical sampling, retry timers for communications protocols and noise generation.

Safety (TMU)

Thermal Monitor Unit (TMU)

The [Thermal Monitoring Unit \(TMU\)](#) unit senses the die temperature during runtime. In cases of weak temp violations an alert (interrupt) is given or in case for severe violations a HW reset is asserted to shut down the system.

Analog Subsystem (HADC)

The [Housekeeping ADC \(HADC\)](#) is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters such as voltages, temperatures in the system or for any general-purpose use as well.

System Debug (SCB/SWU/DBG/TAPC/CSPFT/STM)

The [System Debug and Trace Unit \(DBG\)](#) is based on ARM Core Sight technology. CoreSight™ is a set of architecture specifications defining debug and trace architecture. The processor uses CoreSight infrastructure to provide industry standard debug and trace capabilities through the following modules.

- [Test Access Port Controller \(TAPC\)](#)
- [Trace Unit \(CSPFT\)](#)
- [System Trace Module \(STM\)](#)

Additional debug resources are provided by the following modules.

- The [System Crossbars \(SCB\)](#) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable Quality of Service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.
- The [System Watchpoint Unit \(SWU\)](#) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

2 Clock Generation Unit (CGU)

The Clock Generation Unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a master clock that runs at a frequency that is a multiple of the CLKIN input clock frequency. The PCU divides down the master clock to generate various system clocks and synchronization signals.

CGU Features

The CGU module supports the following features:

- Provides smooth transitions from the current clock condition to a new condition with PLL logic and executes the changes to clocks due to register programming.
- Provides PLL and clock domain status reporting for event management.
- Supports the capability to bypass the PLL for power savings.
- Software controlled dynamic power management allows control of the core clock frequency (f_{CLK}).
- Controls clock gating of the core and system clocks.

NOTE: For more information about processor-specific CGU features, see the processor data sheet.

CGU Functional Description

The CGU generates all on-chip clocks and synchronization signals based on the programmed PLL multiplication factor and dividers. The CGU provides the following functionality.

Change the PLL Clock Frequency

The CGU allows programs to change the PLL clock frequency by writing new values to bits in the control register. Any time the PLL rellocks, the CGU aligns all core and system clocks.

Change Other Clock Frequencies

The CGU allows programs to change the CCLK_n, SYSCLK, SCLK_n, DCLK, and OCLK frequencies by writing values to the [CGU_DIV](#) register. Any time the clock frequency is changed, the OCLK, CCLK_n, SYSCLK, DCLK, and SCLK_n clocks exit the frequency change sequence aligned.

Perform Clock Alignment

The CGU can align all clocks by writing to the [CGU_DIV](#) register. This function aligns all PLL-based clocks.

For more information on these functions, see the [CGU Programming Model](#) section.

ADSP-2156x CGU Register List

The clock generation unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock, running at a frequency that is a multiple of the CLKIN input clock's frequency. The CGU also generates all on-chip clocks and synchronization signals. The PCU permits application software control of the PLL's operation. A set of registers govern CGU operations. For more information on CGU functionality, see the CGU register descriptions.

Table 2-1: ADSP-2156x CGU Register List

Name	Description
CGU_CCBF_DIS	Core Clock Buffer Disable Register
CGU_CCBF_STAT	Core Clock Buffer Status Register
CGU_CLKOUTSEL	CLKOUT Select Register
CGU_CTL	Control Register
CGU_DIV	Clocks Divisor Register
CGU_DIVEX	DIV Register Extension
CGU_OSCWDCTL	Oscillator Watchdog Register
CGU_PLLCTL	PLL Control Register
CGU_REVID	Revision ID Register
CGU_SCBF_DIS	System Clock Buffer Disable Register
CGU_SCBF_STAT	System Clock Buffer Status Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Time Stamp Control Register
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

ADSP-2156x CGU Interrupt List

Table 2-2: ADSP-2156x CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
1	CGU0_EVT	CGU0 Event	Edge	
2	CGU1_EVT	CGU1 Event	Edge	

ADSP-2156x CGU Trigger List

Table 2-3: ADSP-2156x CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	CGU0_EVT	CGU0 Event	Edge
2	CGU1_EVT	CGU1 Event	Edge

Table 2-4: ADSP-2156x CGU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CGU Definitions

DPM

The Dynamic Power Management module (DPM) works with the CGU to provide flexible power dissipation modes for the processor.

PCU

The PLL control unit (PCU) in the CGU controls PLL operations. The MMR registers of the CGU are implemented in the PCU.

PLL

The phase-locked loop (PLL) operates within the CGU.

RCU

The reset control unit (RCU) provides input to the CGU to manage clocks during processor reset.

CDU

The clock distribution unit distributes the clocks from the CGU to different clock domains

CGU

The clock generation unit (CGU) is comprised of the PLL and PCU. The CGU generates the clocks listed in the *Clock Descriptions* table.

Table 2-5: Clock Descriptions

Clock	Description
CCLK0_0	CCLK0 derived from CGU0
SYSCLK_0	SYSCLK derived from CGU0
SCLK0_0	SCLK0 derived from CGU0
SCLK1_0	SCLK1 derived from CGU0
DCLK_0	DCLK derived from CGU0
OCLK_0	OCLK derived from CGU0
DCLK_1	DCLK derived from CGU1

CGU PLL Block Diagram

The *CGU PLL Block Diagram* provides a top-level block diagram of the phase locked loop (PLL). The main blocks of the PLL are the phase/frequency detector (PFD), the charge pump, the loop filter and the voltage controlled oscillator (VCO). The VCO multiplies the `SYS_CLKIN0` input to a higher frequency.

Additional configuration options are configured using the Clock Distribution Unit.

NOTE: The processor supports two PLLs (CGU1–0), see the *CDU Block Diagram* figure in the Clock Distribution (CDU) chapter. Configuring CGU0 is mandatory as it provides the SYSCLK to the system fabric and SCLK0–1 to various peripherals and the SHARC core clock. Configuring CGU1 is optional since it is only used to provide the DCLK, which can also be selected between CGU0 and CGU1.

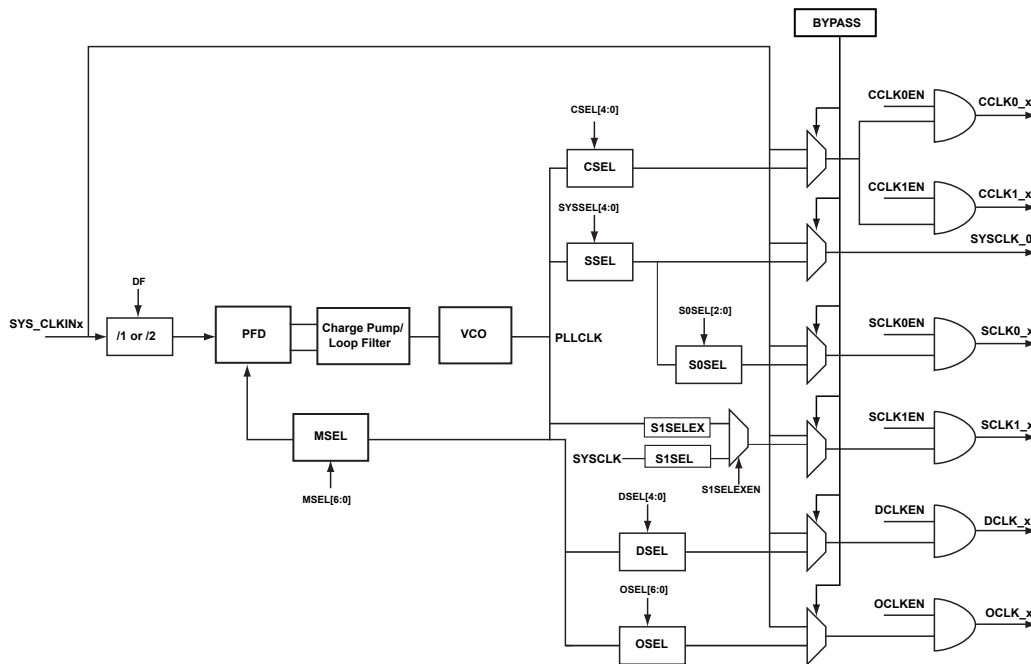


Figure 2-1: CGU PLL Block Diagram

CLKOUT Selections

The *SYS_CLKOUT Generation* figure is a conceptual representation of the CLKOUT module. Different clocks that originate from the CGU blocks are available on the SYS_CLKOUT output pin. The selection of the clock output on the SYS_CLKOUT pin is controlled by the CGU_CLKOUTSEL.CLKOUTSEL bit field. Most of the clock frequencies are divided with a fixed divider value, for example CGU0_SYSCLK0 has a fixed divider of 8 for the CLKOUT pin.

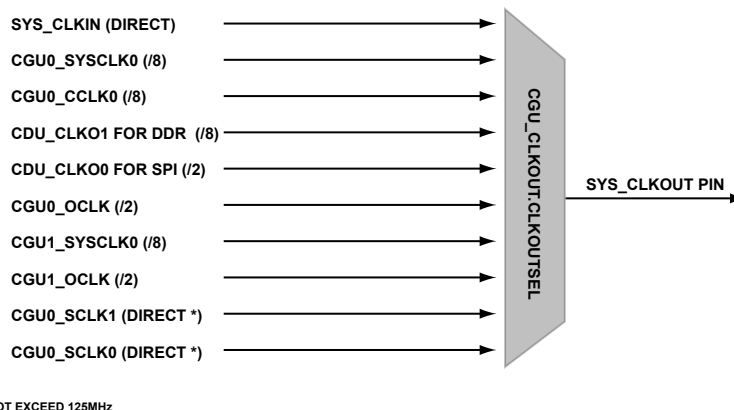


Figure 2-2: SYS_CLKOUT Generation

The BMODE pin affects the CLKOUT pins after a hardware reset is deasserted.

BMODE= 0 — When a hardware reset is deasserted, CGU0_OCLK is selected to come out of the CLKOUT pin. While programming all values of the CGU_CLKOUTSEL.CLKOUTSEL bit field is possible, changing it to 0 (SYS_CLKIN) is not possible until and unless a non-zero value is first programmed to the the CGU_CLKOUTSEL.CLKOUTSEL bit field.

BMODE = (non zero) — When a hardware reset is deasserted, SYS_CLKIN is selected by default.

CGU Operating Modes

The CGU does not have configurable operating modes, but CGU operations affect the operating modes of other modules. Some CGU operation issues that affect operation of other modules include the following:

- The PLL of the CGU operates in either normal mode (CGU clock divisors applied) or bypass mode (CGU PLL is bypassed and clock divisors ignored).
- The SCB uses the CGU for clock synchronization across clock domains. For more information, see [System Crossbars \(SCB\)](#).
- The DPM uses the CGU for clock management as power state transitions occur. For more information, see the [Dynamic Power Management \(DPM\)](#) chapter.
- The CGU uses clock gating control to obtain flexible low-power modes.

CGU Power-up Sequence

See the product data sheet for exact power-up requirements. The processor is configured to come up in clock bypass mode. The programs is required to configure full speed clocks and safety monitors. CLKIN0 and all supplies should be stable before the SYS_HWRST signal is deasserted.

CGU Event Control

The CGU generates an event or error for several different reasons. Events and errors are described in the following sections.

Events

After a frequency change, a CGU event indicates that the PLL has locked and clocks are synchronized. If a core was idled while changing frequencies, the CGU can use an event interrupt to break the core idle. While in active mode, a CGU event indicates that the PLL has locked.

CGU Errors

A CGU error occurs under following conditions:

- A write access to the CGU_DIV/CGU_DIVEX registers trigger an alignment sequence while the PLL is locked and is still aligning the clocks.

The `CGU_STAT.WDIVERR` bit state indicates this error. If this error occurs, clear the `CGU_STAT.WDIVERR` bit and rewrite the desired values to the `CGU_DIV/CGU_DIVEX` registers.

- A change to the `CGU_DIV/CGU_DIVEX` registers occur while the PLL is locked and is still aligning the clocks.

The `CGU_STAT.WDIVERR` bit state indicates this error. If this error occurs, clear the `CGU_STAT.WDIVERR` bit and rewrite the desired values to the `CGU_DIV/CGU_DIVEX` registers.

- A write access to the `CGU_CTL.DF` bit field occurs or a write access to the `CGU_CTL.MSEL` bit field occurs while the PLL is locking.

The `CGU_STAT.WDFMSERR` bit state indicates this error. If this error occurs, wait until the PLL has finished locking, clear the error, and rewrite the desired value change.

The CGU monitors changes to the following fields:

- SYCLK Divisor – `CGU_DIV.SYSSEL`

CGU Generated Bus Errors

The CGU generates a bus error when a read or write transaction to an unused address within the CGU address range is attempted. It also generates a bus error when a misaligned access is made to a CGU register. In addition to the bus error, the `CGU_STAT.ADDRERR` bit is set. If a write to a write-protected CGU register is attempted, the CGU generates a bus error. In addition, the `CGU_STAT.LWERR` bit is set.

Oscillator Watchdog

The oscillator watchdog detects the absence of input clock transitions and provides a fault warning through the `SYS_FAULT` pin. To detect harmonic or subharmonic crystal oscillator behavior, the watchdog (under programmable control) also detects and reports input oscillator frequencies above and below the specified limits. Use an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters for this detection. Set the `CGU_OSCWDCTL.MONDIS` bit and clear the `CGU_OSCWDCTL.FAULTEN` bit to optionally disable all the input clock monitor and fault detection functions.

Set the `CGU_OSCWDCTL.HODEN` bit to enable harmonic oscillation detection. The CGU uses the `CGU_OSCWDCTL.HODF` bit field to indicate the desired lower fail limit (in MHz) for the harmonic oscillation detection. The upper limit is always twice the lower limit.

The *HODF Settings for Different Input Clock Frequencies* table shows the recommended values of the `CGU_OSCWDCTL.HODF` bit settings for different input clock frequencies.

Table 2-6: HODF Settings for Different Input Clock Frequencies

<code>CGU_OSCWDCTL.HODF[5:0]</code>	Subharmonic Frequency (MHz)	Nom. Lower Fail Limit (MHz)	Input Clock Frequency (MHz)	Nom. Upper Fail Limit (MHz)	Second Harmonic Frequency (MHz)
16	10	16	20	32	40

Table 2-6: HODF Settings for Different Input Clock Frequencies (Continued)

CGU_OSCWDCTL.HODF[5:0]	Subharmonic Frequency (MHz)	Nom. Lower Fail Limit (MHz)	Input Clock Frequency (MHz)	Nom. Upper Fail Limit (MHz)	Second Harmonic Frequency (MHz)
16	12.5	16	25	32	50
20	15	20	30	40	60

NOTE: Use a HODF value of 16 for frequency ranges greater than 20 MHz and less than 25 MHz. For frequency ranges greater than 25 MHz and less than 30 MHz, use a HODF value of 20.

The CGU uses the CGU_OSCWDCTL.BOUF (Bad Oscillator Upper Frequency limit) asynchronous control bit field to indicate the desired upper fail limit for the bad oscillation detection. Set the CGU_OSCWDCTL.BOUEN bit to enable upper-limit bad oscillation detection. A bad oscillation detection condition signals a fault before any processor operations occur. This detection occurs (even in bypass mode) whenever a clock frequency exceeds the specifications.

The CGU_OSCWDCTL.BOUF =0 operation starts with a target of 32 MHz and each additional LSB increases the frequency test limit by 2 MHz. For example:

Target Upper Frequency Limit = CGU_OSCWDCTL.BOUF × 2 MHz + 32 MHz

The CGU_STAT.OSCWDSTATFC status bits indicate the nature of the fault.

The *Fault Map* table shows the fault values.

Table 2-7: Fault Map

CGU_STAT.OSCWDSTATFC Bitfield Values	Fault Type
0	No Fault
1	No Input Clock
2	Subharmonic CLKIN
3	Harmonic CLKIN
4	No AUX_CLK
5	CLKIN > Upper Freq Limit (BOUF)
6	Reserved
7	Multiple Limit Faults

There is a priority to the faults given in the case of multiple fault errors. The highest priority is given to No Input clock followed by No AUX_CLK. The other three fault cases share the lowest priority. Multiple limit faults are asserted if more than one type of subharmonic CLKIN, harmonic CLKIN, or BOUF faults are observed.

NOTE: All the CGU_STAT.OSCWDSTATFC faults other than the absence of AUX_CLK (for example, CGU_STAT.OSCWDSTATFC =4) are not reliable and used for debug only.

Program and enable the OSCWDOG to match the actual crystal, before bringing the PLL out of bypass.

CGU Programming Model

The programming model for the CGU involves the various mode configuration techniques.

Configuring CGU Modes

Use the following procedures to configure the clocks and PLL.

NOTE: The program needs to clear the `CGU_STAT.CLKSALGN` bit before changing clocks. The following sequence is executed once, inside the application, after coming out of reset.

```
*pREG_CGU0_PLLCTL |= BITM_CGU_PLLCTL_PLLBPCL; // come out of bypass and enter
Full ON
while( (pADI_CGU0 ->STAT & 0xF) != 0x5 ) { } // poll
// now clocks are running with hardware default divisors.
// now program can change frequencies If desired the program can put the PLL
again into bypass.
```

Changing the Clock Frequencies

Applications change clock frequencies in two ways. The first way is modifying the PLL multiplication value by writing to the `CGU_CTL` register and the second is modifying the clock dividers by writing to the `CGU_DIV` register. Both actions have different implications even if the frequencies of the final clock are the same. Write accesses to change the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit fields while the PLL is locking set the `CGU_STAT.WDFMSERR` error bit. The `CGU_STAT.WDIVERR` error bit is set when one of following accesses is attempted while the PLL is locked, but still aligning the clocks:

- A write access to the `CGU_DIV/CGU_DIVEX` to trigger an alignment sequence.
- A write access to the `CGU_DIV/CGU_DIVEX` to change the `CGU_DIV.CSEL/`, `CGU_DIV.SYSSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.S1SEL`, or `CGU_DIV.DSEL/CGU_DIVEX.S1SELEX` bits.

Read-after-write accesses to these registers return the new value, even if the frequency of the clock change is still in-progress.

Modifying the PLL multiplier requires the PLL to relock. Once the PLL locks, the CGU synchronizes the clocks. Changes to the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit field result in bypassing the PLL. By setting the `CGU_CTL.WFI` bit, programs force the PLL to wait for the core to return to the idle or reset state before the frequency changes. If necessary, clear the `CGU_DIV.UPDT` bit to avoid multiple clock alignment sequences. If the `CGU_DIV` register is not updated, the CGU uses the current values to determine the frequencies of the clock. It is the programs responsibility to guarantee that the new `CGU_CTL.DF` or `CGU_CTL.MSEL` and `CGU_DIV/CGU_DIVEX` combinations are legal.

Changing the PLL Clock Frequency

To change the phase-locked loop clock (*PLLCLK*) frequency, write new values to the `CGU_CTL.MSEL` field or `CGU_CTL.DF` field. Any time the PLL relocks, all core and system clocks are aligned.

1. Read `CGU_STAT` register and verify that:
 - a. The `CGU_STAT.PLLEN` bit =1 (PLL enabled)
 - b. The `CGU_STAT.PLOCK` bit =1 (PLL is not locking)
 - c. The `CGU_STAT.CLKSALGN` bit =0 (clocks aligned)
2. Write the desired values to the clock divisor select fields of the `CGU_DIV/CGU_DIVEX` register with the `CGU_DIV.UPDT` bit =0.
3. Write the desired values to the `CGU_CTL.DF` and `CGU_CTL.MSEL` fields.
 - a. To change the PLL frequency while the core is idle, write to the `CGU_CTL` register with the `CGU_CTL.WFI` bit =1.
 - b. To change the PLL frequency while the core is active, write to the `CGU_CTL` register with the `CGU_CTL.WFI` bit =0.

This sequence performs these actions:

1. Updates the corresponding CGU registers.
2. Bypasses the PLL.
3. Makes the PLL lock to the new values in the `CGU_CTL.MSEL` or `CGU_CTL.DF` fields.
4. Changes the clock frequencies.
5. Exits the PLL bypass with all clocks aligned.

When exiting the PLL bypass state, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.PLLEN` bit =1, the `CGU_STAT.PLOCK` bit =1, the `CGU_STAT.PLLBP` bit =0, and the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.PLOCK` bit, `CGU_STAT.PLLBP` bit, and `CGU_STAT.CLKSALGN` bit to discover when the PLL is locked and the clocks are aligned.

Changing the frequency of the PLL is allowed while the PLL is bypassed. In this case the new PLLCLK frequency is not used until the PLL is no longer bypassed.

Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency

To change the clock frequencies, write new values to `CGU_DIV.CSEL` or `CGU_DIV.SYSSEL` bits. The frequency change occurs only when the PLL is not bypassed. Any time the CCLKn or SYSCLK clock frequencies are changed, they exit the frequency change sequence aligned.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired `CGU_DIV.CSEL`, `CGU_DIV.SYSSEL`, and `CGU_DIV.OSEL` bitfield values with the `CGU_DIV.UPDT` bit = 1 and/or, write to the `CGU_DIVEX.S1SELEX` field. This write updates the

`CGU_DIV` register, changes the `SCLKn` and `SYCLK` frequencies, and aligns the clocks. When the clocks are aligned, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write attempt to change the `CGU_DIV.S0SEL` or `CGU_DIV.S1SEL` bit fields while `CGU_STAT.CLKSALGN` bit =1 (clocks alignment in progress) triggers an MMR access bus error and the `CGU_DIV` register is not modified.

Also, any write attempt to change the `CGU_DIVEX.S1SELEX` field while the `CGU_STAT.CLKSALGN` bit =1 (clocks alignment in progress) triggers an MMR access bus error and the `CGU_DIVEX` register is not modified.

Programming the `SYCLK` frequency to a higher value than `CCLKn` also triggers an MMR access bus error and the `CGU_DIV/CGU_DIVEX` register is not modified.

Writing to the `CGU_DIV/CGU_DIVEX` register is allowed while the processor is in active (PLL bypassed) mode. In this case the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Accessing the DDR memory while changing the `SYCLK` frequency is not supported and can have unpredictable results.

Changing the OCLK Frequency

To change the OCLK clock frequency, write a new `CGU_DIV.OSEL` bit value. Any time the OCLK clock frequency is changed, the OCLK, `CCLKn`, `SYCLK`, and `SCLKn` clocks exit the frequency change sequence aligned.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired `CGU_DIV.OSEL` value with the `CGU_DIV.UPDT` bit =1. This write updates the `CGU_DIV` register, changes the OCLK frequency, and aligns all clocks except OCLK.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write attempt to change the `CGU_DIV.DSEL` field while the `CGU_STAT.CLKSALGN` bit =1 (clock alignment in progress) triggers an MMR access bus error and the `CGU_DIV/CGU_DIVEX` is not modified. When the clocks are aligned, a CGU event occurs.

Writing to the `CGU_DIV.OSEL` bit field is allowed while the processor is in active (PLL bypassed) mode. In this case the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Selecting SCLK1_0 Source

`SCLK1_0` sources the SPDIF receiver clock. This can be sourced from PLL clock or `SYCLK0_0`. Programming the `CGU_DIVEX` enables PLL clock as source (bypassing `SYCLK0_0` that continues to source other peripherals). When `CGU_DIVEX` is enabled, the PLL clock is divided from `CGU_DIVEX` register value.

Changing SCLK1_0 Frequencies

Complete the following steps to change the SCLK0 or SCLK1 frequency.

1. Read the `CGU_STAT` register. Verify that `CGU_STAT.CLKSALGN = 0` (Clocks aligned).
2. Read the `CGU_CTL` register. Verify that the `CGU_CTL.S1SELEXEN` bit is set as needed. (These bits can only be modified when the PLL is bypassed).
3. Read the `CGU_STAT` register. Verify that the PLL is not bypassed. (If it is bypassed, see [PLL Bypass and PLL Disable](#) topic.)
4. Read the `CGU_DIV` register. Verify that the `CGU_DIV.UPDT` bit is set.
5. Write the desired `CGU_DIVEX.S1SELEX` value to the `CGU_DIVEX` registers .

Aligning All Clocks

To align the clocks, write 1 to the `CGU_DIV.ALGN` bit. The frequency can also be changed, if necessary. The clocks aligned include:

- CCLK_n
- SYSCLK
- SCLK_n
- DCLK
- OCLK

1. Read the `CGU_STAT` register to verify that `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write 1 to the `CGU_DIV.ALGN` bit. All other fields can change.

ADDITIONAL INFORMATION: This write does not alter the `CGU_DIV/CGU_DIVEX` register unless one of the clock-select fields is modified. When the clocks are aligned, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write to the `CGU_DIV/CGU_DIVEX` register intended to align clocks or to change a clock select field while the `CGU_STAT.CLKSALGN` bit =1 (clocks alignment in progress) triggers an MMR access bus error. In this case, the `CGU_DIV/CGU_DIVEX` register is not modified.

Writing 1 to the `CGU_DIV.ALGN` bit has no effect while the processor is in active (PLL bypassed) mode.

Shutting Off CCLKn from Another Master

CCLKn can be shut off to save power when it is not in use. In the ADSP-2156x processor, this master is a memory DMA.

1. Disable interrupts to core n.
2. Set the `RCU_SIDIS.SI[n]` bit to disable the interfaces of core n in order to:
 - a. Stop DMA accesses to its L1.
 - b. Stop memory accesses to core 0.
 - c. Stop accesses to MMRs.
3. Test the `RCU_SISTAT.SI[n]` bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
4. Set the `CGU_CCBF_DIS.CCBF0` bit to disable the CCLKn buffer.
5. Check the `CGU_CCBF_STAT.CCBF0` bit.

If the `CGU_CCBF_STAT.CCBF0` bit is set, continue.

Reenable CCLKn From Another Master

1. Clear the `CGU_CCBF_DIS.CCBF0` to enable CCLKn.
2. Check the `CGU_CCBF_STAT.CCBF0` bit.
 - a. If the `CGU_CCBF_DIS.CCBF0` bit is cleared, continue.
3. Clear the `RCU_SIDIS.SI[n]` bit. The core deasserts its acknowledge signal in response to the `RCU_SYSRST0` signal. This operation clears the `RCU_SISTAT.SI[n]` bit.

Valid Clock Multiplier Settings

Processor operations depend on valid settings in the `CGU_CTL` and `CGU_DIV` registers. These registers control the clock multiplier and divisor values. Set these registers such that the minimum and maximum clocks specified in the data sheet are not violated. All other clock specifications in the data sheet must also be adhered to for correct operation of the processor.

PLL Bypass and PLL Disable

Writing 1 to the `CGU_PLLCTL.PLLBPST` bit tells the PLL to apply `OSC_CLKIN` clock to `CCLK`, `SYSCLK`, `SCLK0`, `SCLK1`, `DCLK` (PLL Bypass), and `OCLK` outputs. Writing 1 to the `CGU_PLLCTL.PLLBPCL` bit tells the PLL to exit its PLL Bypass state and make all output clocks align and transition to their programmed frequencies.

The PLL can be disabled by clearing the `CGU_PLLCTL.PLLEN` bit while in the bypass state. If necessary, clock buffers can be disabled.

CCLK0 clock can be disabled or enabled by writing 1 or 0 to the corresponding bit in the `CGU_CCBF_DIS` register.

To determine which core clock buffers were disabled since the last read, software reads the `CGU_CCBF_STAT` register. The SCLK0, SCLK1, DCLK and OCLK clocks can be disabled or enabled by writing 1 or 0 to the corresponding bit in the `CGU_SCBF_DIS` register. Software cannot disable SYSCLK.

ADSP-2156x Specific Information

The processor has one system crystal oscillator and two system CGU units to provide the clocks to the system. Both of the CGUs come up in bypass mode out of reset.

CGU0 is the main CGU which provides most of the clocks including CCLK0, SYSCLK (SYSCLK_0), SCLK0 (SCLK0_0), and SCLK1 (SCLK1_0) to the system buses, infrastructure, and most of the peripherals. The DCLK (DDR clock) source can be selected between CGU0 and CGU1, through CDU0. For details, refer to the [Clock Distribution Unit \(CDU\)](#).

NOTE: The frequency ratio of the core clock to SYSCLK is 2:1.

The frequency ratios of SYSCLK to SCLK0 are 6:1, 4:1 or 2:1.

The processor does not support any other frequency ratio.

ADSP-2156x CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 2-8: ADSP-2156x CGU Register List

Name	Description
<code>CGU_CCBF_DIS</code>	Core Clock Buffer Disable Register
<code>CGU_CCBF_STAT</code>	Core Clock Buffer Status Register
<code>CGU_CLKOUTSEL</code>	CLKOUT Select Register
<code>CGU_CTL</code>	Control Register
<code>CGU_DIV</code>	Clocks Divisor Register
<code>CGU_DIVEX</code>	DIV Register Extension
<code>CGU_OSCWDCTL</code>	Oscillator Watchdog Register
<code>CGU_PLLCTL</code>	PLL Control Register
<code>CGU_REVID</code>	Revision ID Register
<code>CGU_SCBF_DIS</code>	System Clock Buffer Disable Register
<code>CGU_SCBF_STAT</code>	System Clock Buffer Status Register
<code>CGU_STAT</code>	Status Register
<code>CGU_TSCOUNT0</code>	Time Stamp Counter 32 LSB Register

Table 2-8: ADSP-2156x CGU Register List (Continued)

Name	Description
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Time Stamp Control Register
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

Core Clock Buffer Disable Register

The `CGU_CCBF_DIS` register controls each core's clock buffer to determine if the CCLK is enabled.

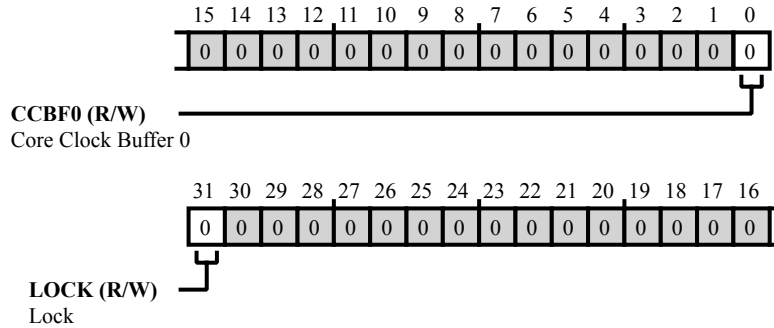


Figure 2-3: `CGU_CCBF_DIS` Register Diagram

Table 2-9: `CGU_CCBF_DIS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If set (=1) the <code>CGU_CCBF_DIS . LOCK</code> bit locks the <code>CGU_CCBF_DIS</code> register.
		0 Unlock register 1 Lock register
0 (R/W)	CCBF0	Core Clock Buffer 0.
		The <code>CGU_CCBF_DIS . CCBF0</code> bit enables (=0) or disables (=1) CCLK0s buffer.
		0 Enable buffer 1 Disable buffer

Core Clock Buffer Status Register

The `CGU_CCBF_STAT` register shows which core clock buffer(s) are disabled. For example clearing the `CGU_CCBF_DIS.CCBF0` bit clears the `CGU_CCBF_STAT.CCBF0` bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

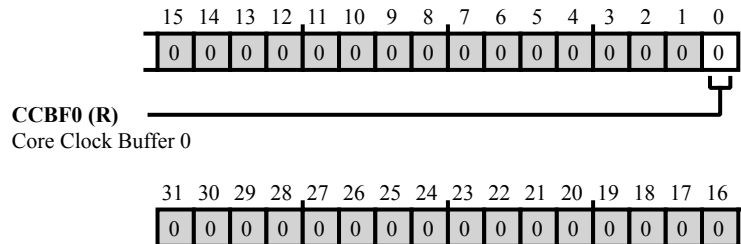


Figure 2-4: `CGU_CCBF_STAT` Register Diagram

Table 2-10: `CGU_CCBF_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	CCBF0	Core Clock Buffer 0. The <code>CGU_CCBF_STAT.CCBF0</code> bit reports the status of the <code>CGU_CCBF_DIS.CCBF0</code> bit where 0 = enabled and 1 = disabled.
		0 Enabled
		1 Disabled

CLKOUT Select Register

The `CGU_CLKOUTSEL` selects the signal that the CGU drives through the CLKOUT multiplexer. Also, this register selects the divisor for the USBCLK output.

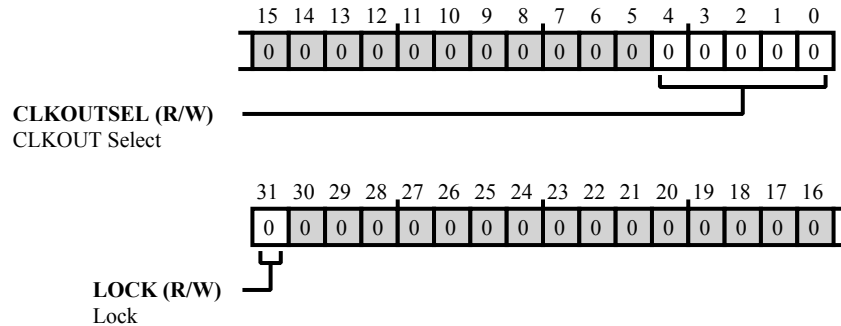


Figure 2-5: CGU_CLKOUTSEL Register Diagram

Table 2-11: CGU_CLKOUTSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>CGU_CLKOUTSEL.LOCK</code> bit is set, the <code>CGU_CLKOUTSEL</code> register is read only (locked).
		0 Unlock
		1 Lock
4:0 (R/W)	CLKOUTSEL	CLKOUT Select. The <code>CGU_CLKOUTSEL.CLKOUTSEL</code> selects the signal that the CGU drives through the CLKOUT pin multiplexer.
		0 CLKIN0
		1 GND
		2 CGU0.SYSCLK0
		3 CGU0.CCLK0
		4 CDU.CLKO1 for DDR
		5 CDU.CLKO0 for SPI
		6 CGU0.OCLK
		7 CGU1.SYSCLK0
		8 CGU1.OCLK
9 Reserved		

Table 2-11: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Reserved
		11	Reserved
		12	Reserved
		13	Reserved
		14	CGU0.SCLK1
		15	Reserved
		16	CGU0.SCLK0
		17	Reserved
		18	Reserved

Control Register

The `CGU_CTL` controls the clock generation divisors for `SYS_CLKIN` and the PLL. Read after write accesses to the `CGU_CTL` register returns the new value even if the clock's frequency change is still in progress.

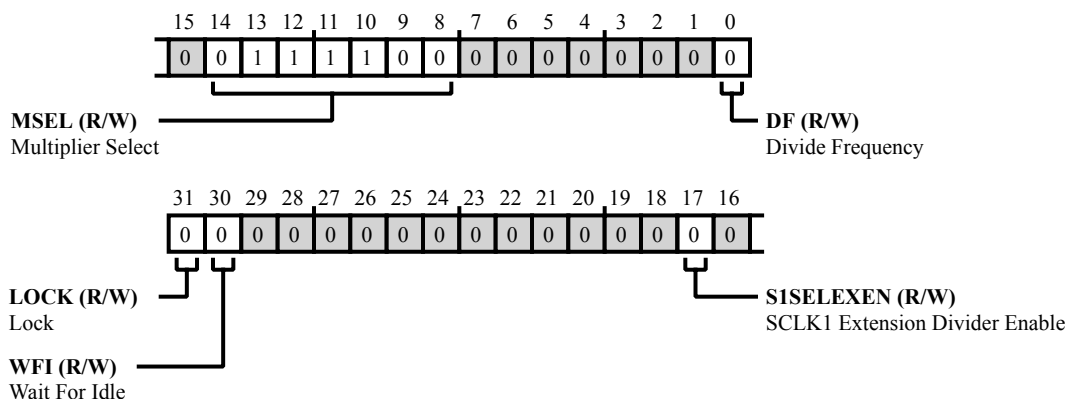


Figure 2-6: CGU_CTL Register Diagram

Table 2-12: CGU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL . GLCK</code> bit =1) and the <code>CGU_CTL . LOCK</code> bit is set, the <code>CGU_CTL</code> register is read only (locked).
		0 Unlock
		1 Lock
30 (R/W)	WFI	Wait For Idle. Modifying the PLL multiplier requires the PLL to re-lock and once the PLL locks, clocks have to be synchronized. Changes to the <code>CGU_CTL . MSEL</code> and the <code>CGU_CTL . DF</code> bit values results in bypassing the PLL. The <code>CGU_CTL . WFI</code> bit forces the PLL to wait for all processor cores to be in an idle or reset state before changing frequencies as a result of changes to the <code>CGU_CTL . MSEL</code> or <code>CGU_CTL . DF</code> bits. Write accesses to the <code>CGU_CTL</code> to change the <code>CGU_CTL . DF</code> or <code>CGU_CTL . MSEL</code> bit values while the PLL is locking sets the <code>CGU_STAT . WDFMSERR</code> bit.
		0 Update Immediately
		1 Wait for Idle

Table 2-12: CGU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	S1SELEXEN	SCLK1 Extension Divider Enable. The CGU_CTL.S1SELEXEN directs the CGU to select S1SELEX instead of S1SEL. This bit can only be updated when the PLL is bypassed.
		0 Selects SYSCLK to SCLK1 Divider
		1 Selects PLLCLK to SCLK1 Divider
14:8 (R/W)	MSEL	Multiplier Select. The CGU_CTL.MSEL bit field selects the multiplier in the PLLCLK equation: $PLLCLK \text{ frequency} = (\text{SYS_CLKIN frequency} / (\text{DF}+1)) * \text{MSEL}$ Where the value of MSEL is between 1 and 127. When the processor comes out of reset, the default multiplier value is 60 for 1 GHz /600 MHz CCLK, 40 for 400 MHz CCLK, and 50 for 800 MHz CCLK processor variants.
		0 MSEL = 128
		1-127 MSEL = 1 to 127
0 (R/W)	DF	Divide Frequency. The CGU_CTL.DF bit selects whether or not the CLKIN input is divided by two before being passed to the PLL.
		0 Pass OSC_CLKIN to PLL
		1 Pass OSC_CLKIN/2 to PLL

Clocks Divisor Register

The `CGU_DIV` register controls clock divisors for core clocks, system clocks, external (off core) memory clocks, and output clock. Read after write accesses to the `CGU_DIV` register returns the new value even if the clock's frequency change is still in progress.

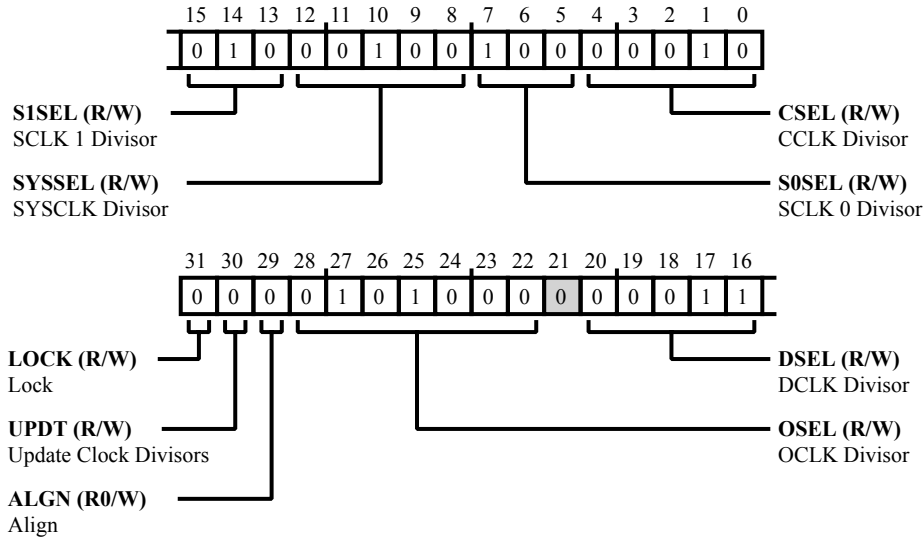


Figure 2-7: CGU_DIV Register Diagram

Table 2-13: CGU_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		0 Unlock
		1 Lock
30 (R/W)	UPDT	Update Clock Divisors.
		The <code>CGU_DIV.UPDT</code> controls whether the CGU drives new <code>CGU_DIV.CSEL</code> , <code>CGU_DIV.SYSEL</code> , <code>CGU_DIV.S0SEL</code> , <code>CGU_DIV.S1SEL</code> , <code>CGU_DIV.DSEL</code> , and <code>CGU_DIV.OSEL</code> values to PLL after <code>CGU_DIV</code> register update.
		0 No PLL Update
		1 Drive Updated SEL Values to PLL

Table 2-13: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R0/W)	ALGN	Align. The CGU_DIV.ALGN directs the CGU to align the PLL-based clocks. The divisor selections (CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and/or CGU_DIV.OSEL) do not have to change.
		0 No Action
		1 Align PLL Clocks
28:22 (R/W)	OSEL	OCLK Divisor. The CGU_DIV.OSEL selects the divisor in the OCLK equation: $OCLK\ frequency = (SYS_CLKIN\ frequency / (DF+1)) * MSEL / CGU_DIV.OSEL$ Where the value of CGU_DIV.OSEL is between 1 and 127.
		0 OSEL = 128
		1-127 OSEL = 1 to 127
20:16 (R/W)	DSEL	DCLK Divisor. The CGU_DIV.DSEL selects the divisor in the DCLK equation: $DCLK\ frequency = (SYS_CLKIN\ frequency / (DF+1)) * MSEL / CGU_DIV.DSEL$ Where the value of CGU_DIV.DSEL is between 1 and 31.
		0 DSEL = 32
		1-31 DSEL = 1 to 31
15:13 (R/W)	S1SEL	SCLK 1 Divisor. The CGU_DIV.S1SEL selects the divisor in the SCLK1 equation: $SCLK1\ frequency = (SYSCLK\ frequency) / CGU_DIV.S1SEL$ Where the value of CGU_DIV.S1SEL is between 1 and 7.
		0 S1SEL = 8
		1-7 S1SEL = 1 to 7
12:8 (R/W)	SYSSEL	SYSCLK Divisor. The CGU_DIV.SYSSEL selects the divisor in the SYSCLK equation: $SYSCLK\ frequency = (SYS_CLKIN\ frequency / (DF+1)) * MSEL / CGU_DIV.SYSSEL$ Where the value of CGU_DIV.SYSSEL is between 1 and 31.
		0 SYSSEL = 32
		1-31 SYSSEL = 1 to 31

Table 2-13: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:5 (R/W)	S0SEL	SCLK 0 Divisor. The CGU_DIV.S0SEL selects the divisor in the SCLK0 equation: $SCLK0 \text{ frequency} = (\text{SYSCLK frequency}) / \text{CGU_DIV.S0SEL}$ Where the value of CGU_DIV.S0SEL is between 1 and 7.
		0 S0SEL = 8
		1-7 S0SEL = 1 to 7
4:0 (R/W)	CSEL	CCLK Divisor. The CGU_DIV.CSEL bit field selects the divisor in the CCLK equation: $CCLK \text{ frequency} = (\text{SYS_CLKIN frequency} / (\text{DF}+1)) * \text{MSEL} / \text{CGU_DIV.CSEL}$ Where the value of CGU_DIV.CSEL is between 1 and 31. When the processor comes out of reset, the default CGU_DIV.CSEL bit field value is 3 for 400/600 MHz CCLK processor variants and 2 for 1GHz/800 MHz CCLK processor variants.
		0 CSEL = 32
		1-31 CSEL= 1 to 31

DIV Register Extension

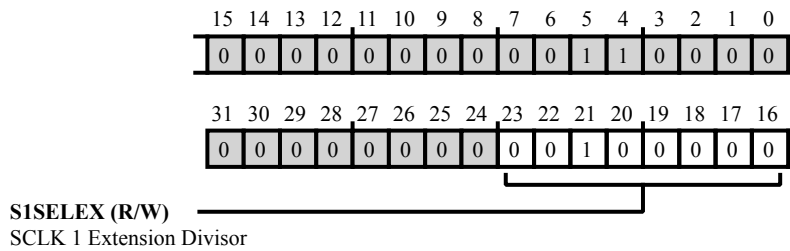


Figure 2-8: CGU_DIVEX Register Diagram

Table 2-14: CGU_DIVEX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	S1SELEX	SCLK 1 Extension Divisor. The CGU_DIVEX.S1SELEX selects the divisor in the SCLK1EX equation: SCLK1EX frequency = (PLLCLK frequency) / CGU_DIVEX.S1SELEX Where the value of CGU_DIVEX.S1SELEX is between 1 and 255.
		0 S1SELEX = 256
		1-255 S1SELEX = 1 to 255

Oscillator Watchdog Register

The `CGU_OSCWDCTL` register configures the CGU to allow the detection of the absence of input clock transitions and provides a fault warning via the `SYS_FAULT` pin. The `CGU_OSCWDCTL` register also detects and reports input oscillator frequencies above and below specified limits, in order to specifically detect harmonic or sub-harmonic crystal oscillator behavior. This detection is achieved by using an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters.

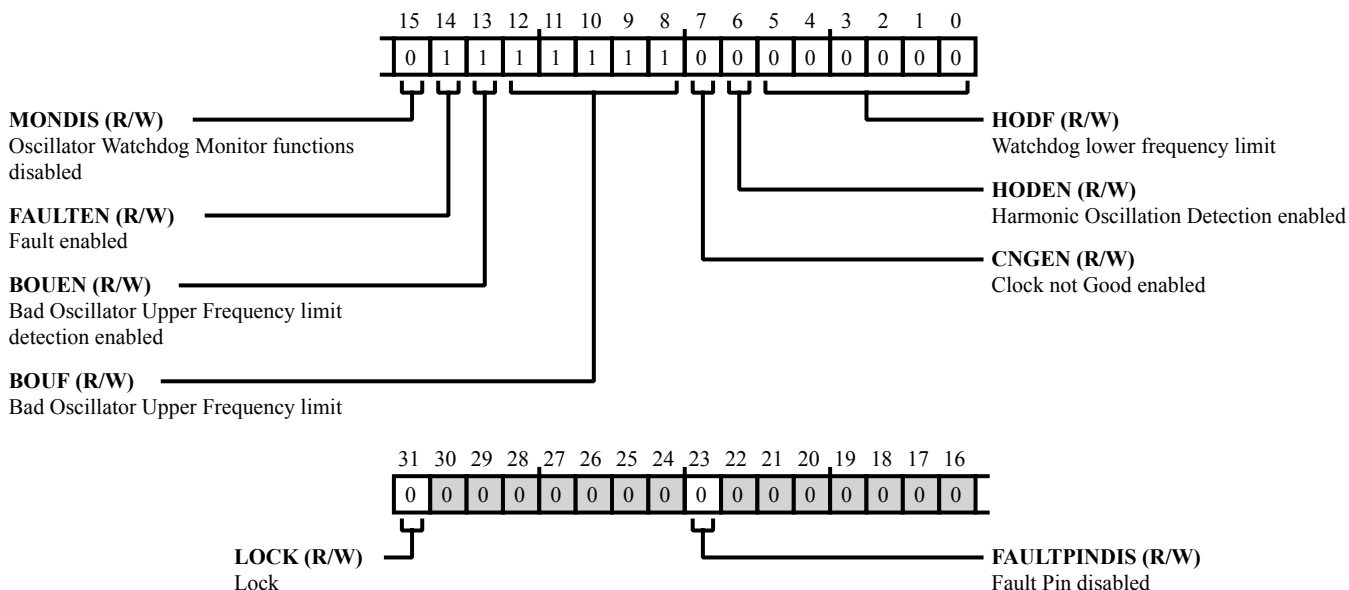


Figure 2-9: `CGU_OSCWDCTL` Register Diagram

Table 2-15: `CGU_OSCWDCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	<code>LOCK</code>	Lock.
23 (R/W)	<code>FAULTPINDIS</code>	Fault Pin disabled. The <code>CGU_OSCWDCTL.FAULTPINDIS</code> bit disables pin fault detection.
15 (R/W)	<code>MONDIS</code>	Oscillator Watchdog Monitor functions disabled. The <code>CGU_OSCWDCTL.MONDIS</code> bit disables all the input clock monitor and fault detection functions.
14 (R/W)	<code>FAULTEN</code>	Fault enabled. The <code>CGU_OSCWDCTL.FAULTEN</code> bit enables fault detection.
13 (R/W)	<code>BOUEN</code>	Bad Oscillator Upper Frequency limit detection enabled. The <code>CGU_OSCWDCTL.BOUEN</code> bit enables upper limit bad oscillation detection.

Table 2-15: CGU_OSCWDCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12:8 (R/W)	BOUF	Bad Oscillator Upper Frequency limit. The CGU_OSCWDCTL.BOUF bits indicate the desired upper fail limit for the bad oscillation detection.
7 (R/W)	CNGEN	Clock not Good enabled. The CGU_OSCWDCTL.CNGEN bit enables the detection of an oscillator watchdog clock fault.
6 (R/W)	HODEN	Harmonic Oscillation Detection enabled. The CGU_OSCWDCTL.HODEN bit enables harmonic oscillation detection.
5:0 (R/W)	HODF	Watchdog lower frequency limit. The CGU_OSCWDCTL.HODF bit field is used to indicate the desired lower fail limit for the harmonic oscillation detection in MHz.

PLL Control Register

The `CGU_PLLCTL` register contains bits that enable and disable the PLL as well as control its function.

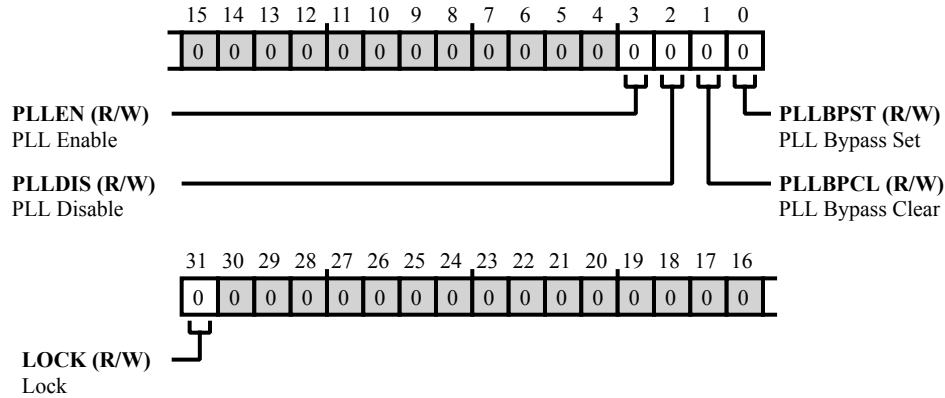


Figure 2-10: `CGU_PLLCTL` Register Diagram

Table 2-16: `CGU_PLLCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. Setting (=1) the <code>CGU_PLLCTL . LOCK</code> bit locks access to the <code>CGU_PLLCTL</code> register.
		0 Unlock register
		1 Lock register
3 (R/W)	PLLEN	PLL Enable. Setting (=1) the <code>CGU_PLLCTL . PLLEN</code> bit enables the PLL. Check the <code>CGU_STAT . PLLEN</code> bit to verify that the action is complete.
		0 No action
		1 Enable PLL
2 (R/W)	PLLDIS	PLL Disable. Setting (=1) the <code>CGU_PLLCTL . PLLDIS</code> bit disables the PLL. Check the <code>CGU_STAT . PLLEN</code> bit to verify that the action is complete.
		0 No action
		1 Disable PLL
1 (R/W)	PLLBPCL	PLL Bypass Clear. Setting (=1) the <code>CGU_PLLCTL . PLLBPCL</code> bit takes the PLL out of bypass mode. Check the <code>CGU_STAT . PLLBP</code> bit to verify that the action is complete.
		0 No action
		1 Exit bypass mode

Table 2-16: CGU_PLLCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PLLBPST	PLL Bypass Set. Setting (=1) the CGU_PLLCTL.PLLBPST bit bypasses the PLL and all the clocks run on CLKIN. Check the CGU_STAT.PLLBP bit to verify that the action is complete.	
		0	Use PLL
		1	Bypass PLL

Revision ID Register

The `CGU_REVID` register reports the version of the CGU.

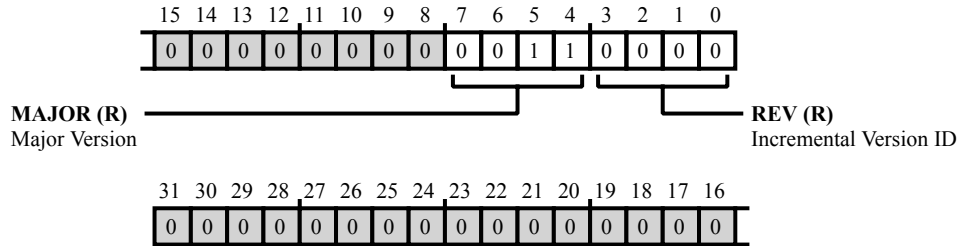


Figure 2-11: `CGU_REVID` Register Diagram

Table 2-17: `CGU_REVID` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version.
3:0 (R/NW)	REV	Incremental Version ID.

System Clock Buffer Disable Register

The `CGU_SCBF_DIS` register controls each system's clock buffer to determine if the `SCLKn` buffer is enabled.

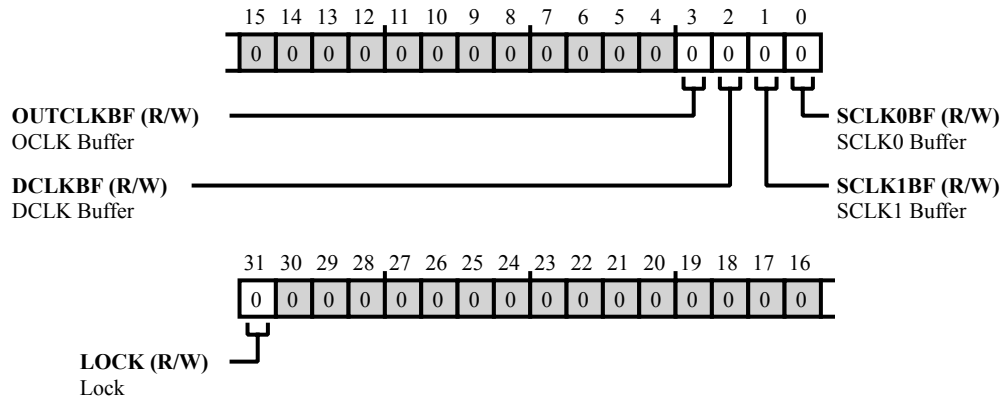


Figure 2-12: `CGU_SCBF_DIS` Register Diagram

Table 2-18: `CGU_SCBF_DIS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		The <code>CGU_SCBF_DIS</code> . <code>LOCK</code> bit allows writes to the <code>CGU_SCBF_DIS</code> register when cleared (=0) or blocks writes if set (=1) and the <code>SPU_CTL</code> . <code>GLCK</code> bit is set.
		0 Unlock register 1 Lock register
3 (R/W)	OUTCLKBF	OCLK Buffer.
		The <code>CGU_SCBF_DIS</code> . <code>OUTCLKBF</code> bit enables (=0, default) or disables (=1) OCLKs buffer.
		0 Enable buffer 1 Disable buffer
2 (R/W)	DCLKBF	DCLK Buffer.
		The <code>CGU_SCBF_DIS</code> . <code>DCLKBF</code> bit enables (=0, default) or disables (=1) DCLKs buffer.
		0 Enable buffer 1 Disable buffer

Table 2-18: CGU_SCBF_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	SCLK1BF	SCLK1 Buffer. The CGU_SCBF_DIS.SCLK1BF bit enables (=0, default) or disables (=1) SCLK1s buffer.
		0 Enable buffer
		1 Disable buffer
0 (R/W)	SCLK0BF	SCLK0 Buffer. The CGU_SCBF_DIS.SCLK0BF bit enables (=0, default) or disables (=1) SCLK0s buffer.
		0 Enable buffer
		1 Disable buffer

System Clock Buffer Status Register

The `CGU_SCBF_STAT` register shows which system clock buffer(s) are disabled. For example clearing the `CGU_CCBF_DIS.CCBF0` bit clears the `CGU_SCBF_STAT.SCLK0BF` bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

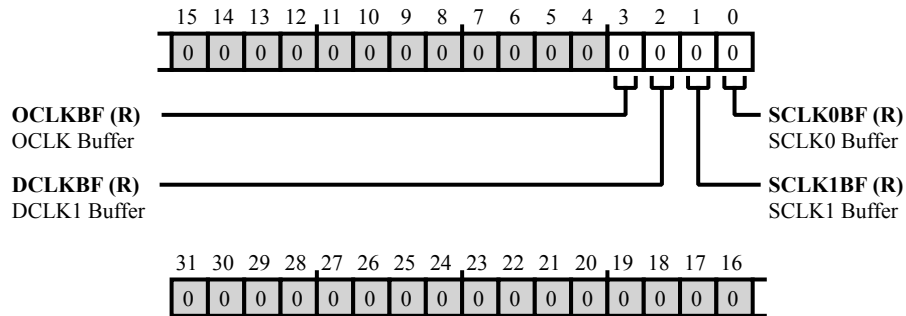


Figure 2-13: `CGU_SCBF_STAT` Register Diagram

Table 2-19: `CGU_SCBF_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	OCLKBF	OCLK Buffer. The <code>CGU_SCBF_STAT.OCLKBF</code> bit reports the status of the <code>CGU_SCBF_DIS.OUTCLKBF</code> bit where 0 = enabled and 1 = disabled.
		0 Enabled
		1 Disabled
2 (R/NW)	DCLKBF	DCLK1 Buffer. The <code>CGU_SCBF_STAT.DCLKBF</code> bit reports the status of the <code>CGU_SCBF_DIS.DCLKBF</code> bit where 0 = enabled and 1 = disabled.
		0 Enabled
		1 Disabled
1 (R/NW)	SCLK1BF	SCLK1 Buffer. The <code>CGU_SCBF_STAT.SCLK1BF</code> bit reports the status of the <code>CGU_SCBF_DIS.SCLK1BF</code> bit where 0 = enabled and 1 = disabled.
		0 Enabled
		1 Disabled

Table 2-19: CGU_SCBF_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SCLK0BF	SCLK0 Buffer. The CGU_SCBF_STAT.SCLK0BF bit reports the status of the CGU_SCBF_DIS.SCLK0BF bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

Status Register

The `CGU_STAT` register reflects the PLL status and errors detected during the PLL configuration. This register may be cleared asynchronously by a reset signal from the RCU module. All bits---except those defined as W1C (write-1-to-clear)---are read only.

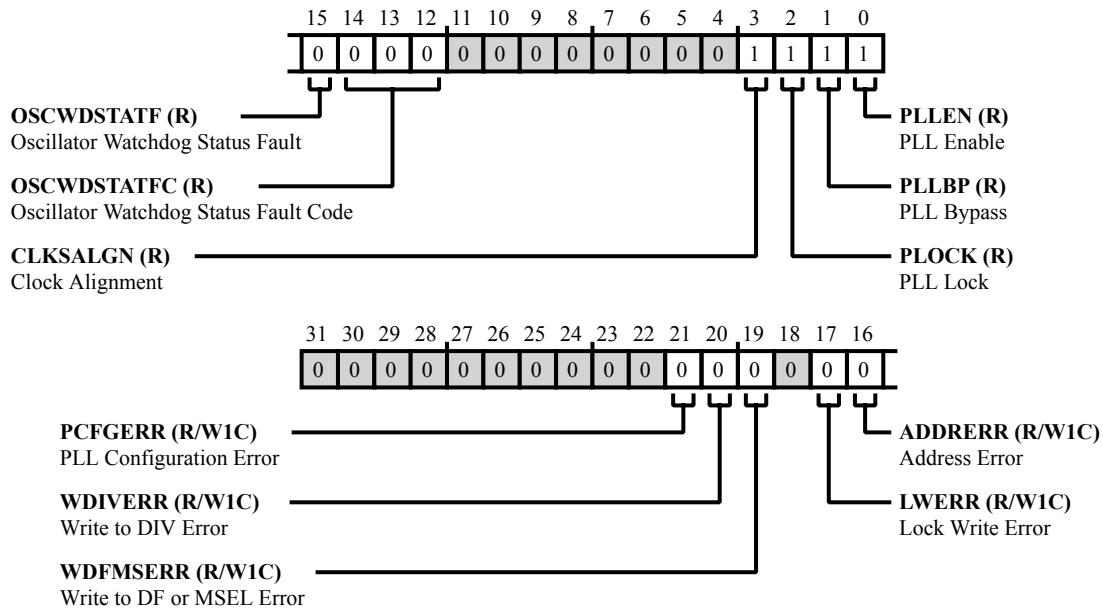


Figure 2-14: CGU_STAT Register Diagram

Table 2-20: CGU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W1C)	PCFGERR	PLL Configuration Error. If the <code>CGU_PLLCTL.PLLBPST</code> and the <code>CGU_PLLCTL.PLLBPCL</code> bits are set (=1) simultaneously or the <code>CGU_PLLCTL.PLLDIS</code> bit was set (=1) in full-on mode or while trying to enter full-on mode (<code>CGU_PLLCTL.PLLBPCL</code> =1), the <code>CGU_STAT.PCFGERR</code> bit triggers the bus error.
		0 No Error
		1 Configuration Error

Table 2-20: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W1C)	WDIVERR	Write to DIV Error. The CGU_STAT.WDIVERR bit indicates a write access to the CGU_DIV register (to trigger an alignment sequence or to change the CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL bit values) while the PLL is locked, but still aligning the clocks. Read after write accesses to the CGU_STAT and CGU_DIV registers return the new value even if the clock frequency change is still in progress.
		0 No Error
		1 Write DIV Error
19 (R/W1C)	WDFMSERR	Write to DF or MSEL Error. The CGU_STAT.WDFMSERR bit indicates a write access to the CGU_CTL register to change the CGU_CTL.DF or CGU_CTL.MSEL bit values while the PLL is locking.
		0 No Error
		1 Write DF/MSEL Error
17 (R/W1C)	LWERR	Lock Write Error. The CGU_STAT.LWERR bit indicates an attempt to write to write-protected (locked) CGU registers. The CGU issues a bus error for this condition.
		0 No Error
		1 Lock Write Error
16 (R/W1C)	ADDRERR	Address Error. The CGU_STAT.ADDRERR bit indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The CGU issues a bus error for this condition.
		0 No Error
		1 Address Error
15 (R/NW)	OSCWDSTATF	Oscillator Watchdog Status Fault. The CGU_STAT.OSCWDSTATF bit indicates a fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.
		0 No Fault
		1 Fault
14:12 (R/NW)	OSCWDSTATFC	Oscillator Watchdog Status Fault Code. The CGU_STAT.OSCWDSTATFC bit field indicates the nature of the fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.
		0 No Fault
		1 No Input Clock

Table 2-20: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		2 Subharmonic CLKIN
		3 Harmonic CLKIN
		4 No AUX_CLK
		5 CLKIN > Upper Frequency Limit (BOUF)
		6 Reserved
		7 Multiple Limit Faults
3 (R/NW)	CLKSALGN	<p>Clock Alignment.</p> <p>The <code>CGU_STAT.CLKSALGN</code> bit indicates whether a clock alignment sequence is in progress. This bit is set when clocks alignment is required by changes to <code>CGU_DIV.CSEL</code>, <code>CGU_DIV.S0SEL</code>, <code>CGU_DIV.S1SEL</code>, <code>CGU_DIV.DSEL</code>, or <code>CGU_DIV.OSEL</code>. The <code>CGU_STAT.CLKSALGN</code> bit is cleared when clocks are aligned.</p> <p>Note that (after a PLL frequency change in active state) the <code>CGU_STAT.CLKSALGN</code> bit may indicate that clocks are not aligned even though the clocks are aligned (all clocks are aligned and running at CLKIN frequency).</p>
		0 Clocks are Aligned
		1 Clocks not Aligned (alignment in progress)
2 (R/NW)	PLOCK	<p>PLL Lock.</p> <p>The <code>CGU_STAT.PLOCK</code> bit indicates whether the PLL is locked. This bit is set when the PLL locks (PLL lock counter end-of-count). The <code>CGU_STAT.PLOCK</code> bit is cleared when requested PLL frequency change (for PLL reset, PLL disable-to-enable transition, or a change to the <code>CGU_CTL.MSEL</code> or <code>CGU_CTL.DF</code> values) is in progress.</p>
		0 PLL not Locked (PLL frequency change in progress)
		1 PLL Locked
1 (R/NW)	PLLBP	<p>PLL Bypass.</p> <p>The <code>CGU_STAT.PLLBP</code> bit indicates whether the PLL is bypassed. The default value for the <code>CGU_STAT.PLLBP</code> bit is determined by the bypass strap pin.</p>
		0 PLL not Bypassed
		1 PLL Bypassed
0 (R/NW)	PLLEN	<p>PLL Enable.</p> <p>The <code>CGU_STAT.PLLEN</code> bit indicates whether the PLL is enabled.</p>
		0 Disabled
		1 Enabled

Time Stamp Counter 32 LSB Register

The `CGU_TSCOUNT0` register address is used to read the CoreSight time stamp counter LSB 32-bit (bits [31:0]) value.

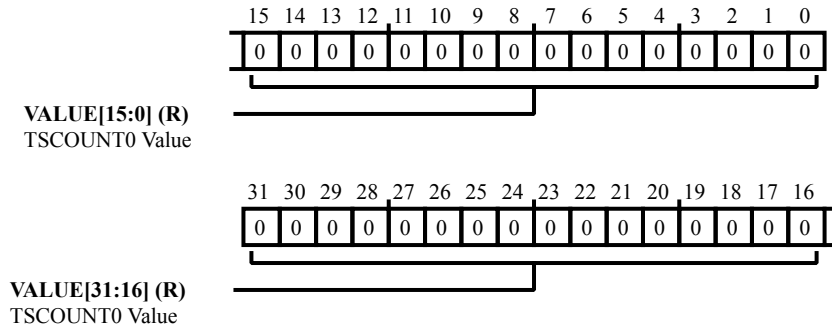


Figure 2-15: `CGU_TSCOUNT0` Register Diagram

Table 2-21: `CGU_TSCOUNT0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	TSCOUNT0 Value. The <code>CGU_TSCOUNT0.VALUE</code> bit field holds the time stamp counter 32 LSBs.

Time Stamp Counter 32 MSB Register

The `CGU_TSCOUNT1` register address is used to read the CoreSight time stamp counter MSB 32-bit (bits [63:32]) value.

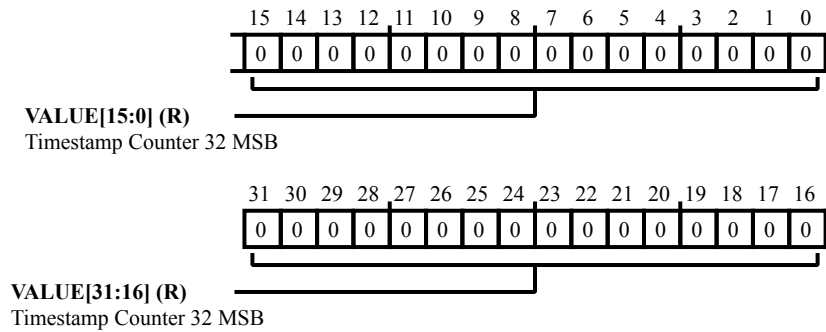


Figure 2-16: `CGU_TSCOUNT1` Register Diagram

Table 2-22: `CGU_TSCOUNT1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Timestamp Counter 32 MSB. The <code>CGU_TSCOUNT1.VALUE</code> bit field holds the time stamp counter 32 MSBs.

Time Stamp Control Register

The `CGU_TSCTL` register controls the operation of the CoreSight time stamp counter.

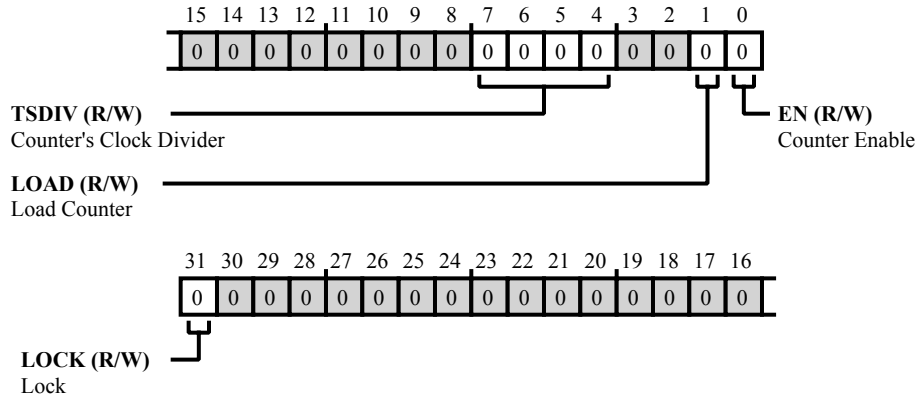


Figure 2-17: `CGU_TSCTL` Register Diagram

Table 2-23: `CGU_TSCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. Setting the <code>CGU_TSCTL.LOCK</code> bit locks this register.
		0 Unlock
		1 Lock
7:4 (R/W)	TSDIV	Counter's Clock Divider. The <code>CGU_TSCTL.TSDIV</code> bit field divides <code>SYSCCLK</code> by 2^{TSDIV} .
		0-15 Divides <code>SYSCCLK</code> by 2^{TSDIV}
1 (R/W)	LOAD	Load Counter. Writing one to the <code>CGU_TSCTL.LOAD</code> bit causes CoreSight time stamp counter to be loaded from the <code>CGU_TSVALUE0</code> and <code>CGU_TSVALUE1</code> registers.
		0 Always read as "0"
0 (R/W)	EN	Counter Enable. The <code>CGU_TSCTL.EN</code> bit enables or disables the CoreSight time stamp counter.
		0 Counter Disabled
		1 Counter Enabled

Time Stamp Counter Initial 32 LSB Value Register

The `CGU_TSVVALUE0` register holds the least significant bits (bits [31:0]) value that is initially loaded to the CoreSight time stamp counter.

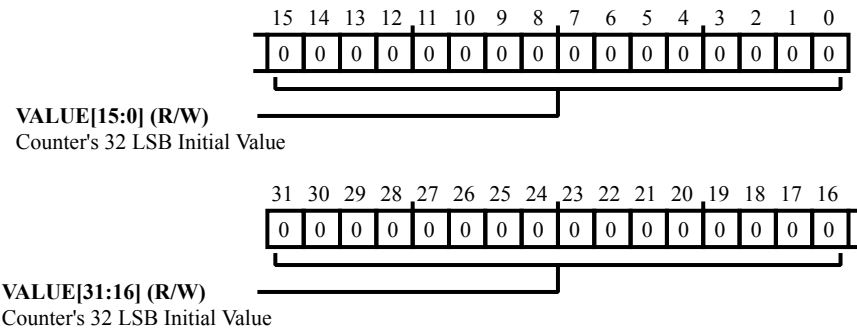


Figure 2-18: `CGU_TSVVALUE0` Register Diagram

Table 2-24: `CGU_TSVVALUE0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's 32 LSB Initial Value. The <code>CGU_TSVVALUE0.VALUE</code> bit field holds the LSBs value that is initially loaded to the CoreSight time stamp counter.

Time Stamp Counter Initial MSB Value Register

The `CGU_TSVVALUE1` register holds the most significant bits (bits [63:32]) value that is initially loaded to the CoreSight time stamp counter.

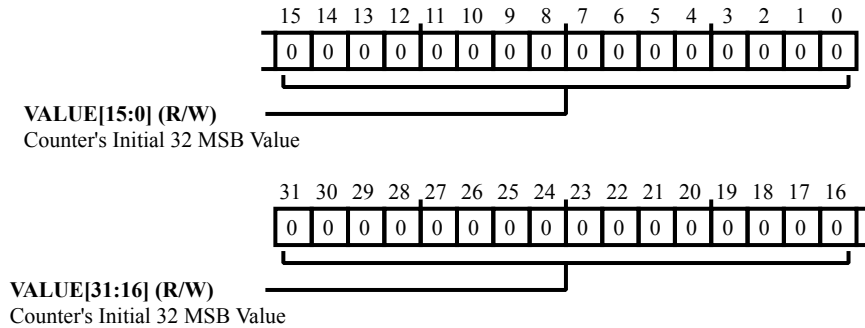


Figure 2-19: `CGU_TSVVALUE1` Register Diagram

Table 2-25: `CGU_TSVVALUE1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's Initial 32 MSB Value. The <code>CGU_TSVVALUE1.VALUE</code> bit field holds the MSBs value that is initially loaded to the CoreSight time stamp counter.

3 Clock Distribution Unit (CDU)

The Clock Distribution Unit (CDU) consists of an array of multiplexors that select clocks originated from up to four different clock sources. These sources are different clocks that are generated from the CGUs. The multiplexors are configured by software. All clocks multiplexors have up to four different sources. Unused input clocks are grounded internally and never selected. The output clock signal for each multiplexor is assigned to one or more destinations within the processor.

CDU Features

The CDU modules supports the following features:

- Generation of up to 2 output clocks
-
- Output clock buffers that can be disabled by software
- Each multiplexor has four input clocks
- Multiplexors that are configured by writing to configuration registers (`CDU_CFG[n]`)
- Clocks originated in CGU0 that are selected by default
- A CDU status register (`CDU_STAT`) that indicates a configuration change is in-progress

CDU Functional Description

The CDU functions as a set of software-configurable multiplexors that select clocks from different sources.

CDU Block Diagram

The *CDU Block Diagram* figure shows the functional blocks within the CDU. As shown in the figure, the CDU takes different clocks generated by the CGU blocks and provides flexibility to route any clock from possible options to the output CDU clocks. The output clocks from the CDU are connected to specific targets such as the DDR module. All of the targets are clocked through the CDU output clocks rather than being directly clocked by CGU clocks.

This configuration provides the flexibility to meet the specific clock requirements of the different modules in the system (such as the core, DDR, or SPI module) without compromising the clocking of other modules. Such a flexibility is not possible with a single CGU in the system. With two CGUs:

- Each CGU has its own set of clock multipliers and dividers, providing a greater number of orthogonal clocks than possible with a single CGU.
- Each CGU is clocked from a single CLKIN source.

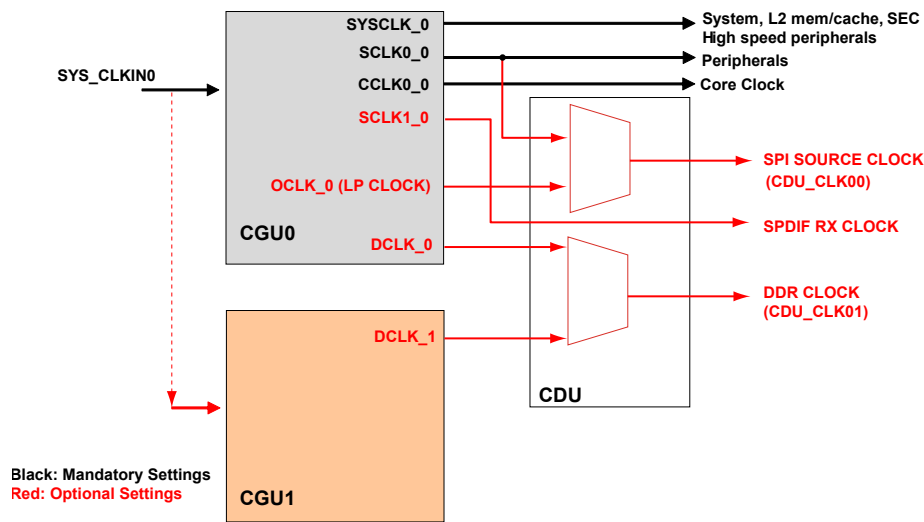


Figure 3-1: CGU/CDU Block Diagram

The DDR is clocked from the DCLK (DCLK_0 and DCLK_1) from CGU0 or CGU1. This configuration provides the flexibility to program the DDR with frequencies orthogonal to the core clock frequencies.

CDU Definitions

The *Clock Descriptions* table provides a brief description of the clocks supported by the processor.

Table 3-1: Clock Descriptions

Clock	Description
CCLK0_0	CCLK0 derived from CGU0
SYSCLK_0	SYSCLK derived from CGU0
SCLK0_0	SCLK0 derived from CGU0
SCLK1_0	SCLK1 derived from CGU0
DCLK_0	DCLK derived from CGU0
OCLK_0	OCLK derived from CGU0
DCLK_1	DCLK derived from CGU1
CDU_CLKOn	Clocks that come out of the CDU module and go to different blocks

CDU Clock Configuration Options

The *CDU Targets* table provides information about clock source and destination options.

Table 3-2: CDU Targets

CDU0 Input		CDU0 Output	Target
IN0_CLKOn	IN1_CLKOn		
SCLK_0	OCLK_0	CLKO0	SPI (0-2) Source clock
DCLK_0	DCLK_1	CLKO1	DDR clock

CDU Programming Model

The `CDU_CFG[n]` registers are a system configuration resource. These registers are accessed by a system configuration routine that also handles the configuration of other system modules. Writes to a `CDU_CFG[n]` register must occur when there is not a `CDU_CLKOn` configuration change in progress. If a CGUn inputs are selected, that CGUn configuration must be completed first.

Changing the PLL and Clock Frequency

1. Read the `CGU_STAT` register. Verify that:
 - `CGU_STAT.PLLEN = 1` (enabled)
 - `CGU_STAT.PLOCK = 1` (PLL is not locking)
 - `CGU_STAT.CLKSALGN = 0` (clocks aligned)
2. Write the desired values into the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bits with the `CGU_DIV.UPDT` bit cleared (= 0).
3. Write the desired values to the `CGU_CTL.DF` and `CGU_CTL.MSEL` bits.
 - To change the PLL frequency while all cores are idle, set the `CGU_CTL.WFI` bit (=1).
 - To change the PLL frequency while the cores are active, clear the `CGU_CTL.WFI` bit (=0).
4. Read the `CGU_STAT` register. Verify that:
 - `CGU_STAT.PLLEN = 1` (enabled)
 - `CGU_STAT.PLOCK = 1` (not locking)
 - `CGU_STAT.CLKSALGN = 0` (clocks aligned)
5. If clocks switch from CGUm clocks to CGUn input clocks, read the `CGU_SCBF_STAT` and `CGU_CCBF_STAT` registers. The `CGU_CCBF_STAT[1:0]` bit field corresponds to the CCLK1 and CCLK0 clock buffers, respectively. The `CGU_SCBF_STAT[3:0]` bit field corresponds to the OUTCLK, DCLK, SCLK1, and SCLK0 clock buffers, respectively.

6. Read the `CDU_STAT` register. Verify that the `CDU_STAT.CLK00` (no CLKOn configuration change in progress).
7. Write to the `CDU_CLKINSEL` register to select the CGU's CLKIN input clock.
8. Write to the `CDU_CFG[n].SEL` bit to select the clock source (the `CDU_CFG[n].EN` bit should =1.)
9. Read the `CDU_CFG[n]` register. Verify that the `CDU_CLKINSEL` has the programmed value.

CLKOn is reconfigured.

Changing the Clock Frequency

Use the following procedure to change a clock frequency.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired values into the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bits with the `CGU_DIV.UPDT` bit = 1.
3. Read the `CGU_DIV` registers to verify that the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bit values are correct.
4. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` = 0 (clocks aligned).
5. If clocks switch from CGUm clocks to CGUn input clocks, read the `CGU_SCBF_STAT` and `CGU_CCBF_STAT` registers. The `CGU_CCBF_STAT[1:0]` bit field corresponds to the CCLK1 and CCLK0 clock buffers, respectively. The `CGU_SCBF_STAT[3:0]` bit field corresponds to the OUTCLK, DCLK, SCLK1, and SCLK0 clock buffers, respectively.
6. Read the `CDU_STAT` register.
7. Write to the `CDU_CFG[n].SEL` bit to select the clock source (`CDU_CFG[n].EN` = 1).
8. Read the `CDU_CFG[n]` register. Verify that the `CDU_CLKINSEL` has the programmed value.
9. Verify that the `CDU_STAT.CLK00` = 0.

ADSP-2156x CDU Register Descriptions

Clock Distribution Unit (CDU) contains the following registers.

Table 3-3: ADSP-2156x CDU Register List

Name	Description
<code>CDU_CFG[n]</code>	CDU Configuration
<code>CDU_CLKINSEL</code>	CLKIN Select
<code>CDU_REVID</code>	CDU Revision ID

Table 3-3: ADSP-2156x CDU Register List (Continued)

Name	Description
CDU_STAT	CDU Status

CDU Configuration

The `CDU_CFG[n]` registers control the configuration of the clock multiplexors. `CDU0_CFG[n]` corresponds to output clock `CDU_CLKO[n]`.

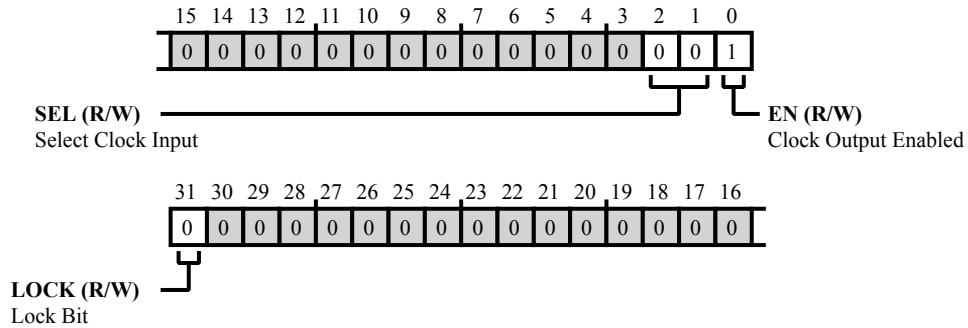


Figure 3-2: `CDU_CFG[n]` Register Diagram

Table 3-4: `CDU_CFG[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
2:1 (R/W)	SEL	Select Clock Input.	
		0	IN0_CLKOn Selected
		1	IN1_CLKOn Selected
		2	IN2_CLKOn Selected
0 (R/W)	EN	Clock Output Enabled. The <code>CDU_CFG[n].EN</code> bit enables clock output.	

CLKIN Select

The `CDU_CLKINSEL` register controls the configuration of the CLKIN multiplexors. One bit is assigned to each CGU in the system. This bit selects either CLKIN0 or CLKINn CGUn inputs.

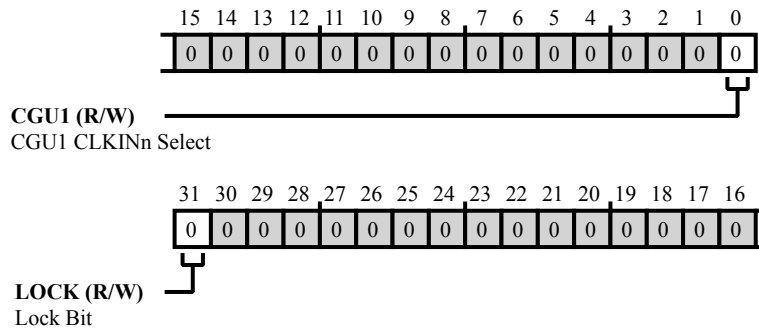


Figure 3-3: `CDU_CLKINSEL` Register Diagram

Table 3-5: `CDU_CLKINSEL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit.
0 (R/W)	CGU1	CGU1 CLKINn Select. The <code>CDU_CLKINSEL.CGU1</code> bit drives <code>CDU_CLKIN_SEL[0]</code> to CGU1.
		0 Selects CLKIN0
		1 Selects CLKIN1

CDU Revision ID

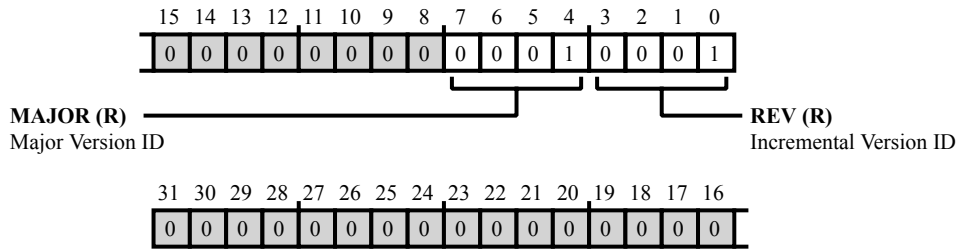


Figure 3-4: CDU_REVID Register Diagram

Table 3-6: CDU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

CDU Status

The `CDU_STAT` register reflects the status a change in the configuration of the clock muxes.

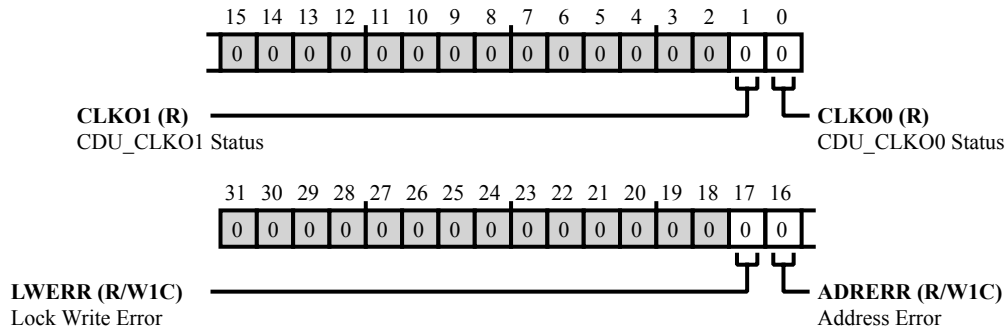


Figure 3-5: `CDU_STAT` Register Diagram

Table 3-7: `CDU_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W1C)	LWERR	Lock Write Error. The <code>CDU_STAT.LWERR</code> bit indicates a lock error where write transactions try to access write protected registers.
		0 No Lock Write Error
		1 Lock Write Error
16 (R/W1C)	ADRERR	Address Error. The <code>CDU_STAT.ADRERR</code> bit indicates an address error where read or write transactions try to access unimplemented addresses or accesses are non-aligned.
		0 No Address Error
		1 Address Error
1 (R/NW)	CLKO1	CDU_CLKO1 Status.
		0 No Configuration Change in Progress
		1 Configuration Change in Progress
0 (R/NW)	CLKO0	CDU_CLKO0 Status.
		0 No Configuration Change in Progress
		1 Configuration Change in Progress

4 Dynamic Power Management (DPM)

The dynamic power management (DPM) unit of the processor controls transitions between different power-saving modes.

DPM Features

The DPM allows programs to control the power mode of the processor as follows.

- Permits operation of multiple, external wake-up sources

DPM Functional Description

The DPM can be programmed to transition between power modes.

ADSP-2156x DPM Register List

A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 4-1: ADSP-2156x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_PER_DIS0	Peripherals Disable Register 0
DPM_PER_DIS1	Peripherals Disable Register 1
DPM_REVID	Revision ID
DPM_STAT	Status Register

DPM Definitions

To make the best use of the DPM, it is useful to understand the following terms.

CGU

Acronym for the clock generation unit (CGU), which is comprised of the PLL and PCU

DPM

Acronym for the dynamic power management (DPM) controller.

Full-on mode

The normal operating mode in which all clock domains are derived from the PLL.

PCU

Acronym for the PLL control unit (PCU).

PLL

Acronym for the phase-locked loop (PLL).

RCU

Acronym for the reset control unit (RCU).

DPM Operating Modes

The DPM includes several operating modes. The modes are:

- Reset
- Full-on

Reset State

Reset is the initial state of the processor and is the result of a hardware or software triggered event. The DPM itself does not trigger entering reset. The external `SYS_HWRST` pin or the RCU triggers entering reset. The DPM responds to reset by transitioning to its default state.

From Reset, the DPM always transitions to PLL Bypassed state.

Full-on Mode

Full-on mode is the default state of the DPM after Reset.

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest.

DPM Event Control

The DPM event is triggered when an enabled wake-up is asserted. The DPM generates bus errors when a misaligned access to a register occurs. It also generates errors when an attempt is made to access unused DPM address space or a write-protected register.

DPM Events

The DPM event interrupt is triggered when any bit in the `DPM_STAT` register is set, indicating that an enabled wake-up was asserted. The DPM event interrupt stays active until the user clears any bits that are set in the `DPM_STAT` register.

DPM Errors

The DPM generates a bus error when a read or write transaction is attempted to an unused address within the DPM address range. It also generates a bus error when a misaligned access is made to a DPM register. In addition to the bus error, the DPM sets the `DPM_STAT.ADDRERR` bit.

If a write to a write-protected DPM register is attempted, the DPM generates a bus error. In addition, the DPM sets the `DPM_STAT.LWERR` bit.

DPM Programming Model

The *DPM_PER_DIS0 Register Mapping* and *DPM_PER_DIS01 Register Mapping* tables show the module clocks and the corresponding peripheral. The `DPM_PER_DIS0` and `DPM_PER_DIS0` registers are used to shut off the clock to each peripheral if it is not required by the application.

Table 4-2: DPM0_PER_DIS0 Register Mapping

Peripheral Name	Gated Module Clocks	Type of sync on PER-DIS bit	Effect latency (In cycles of module clocks)	DPM0_PER_DISn bit
FIR0/FFT	cclk	Not Required	1	0
IIR0	cclk	Not Required	1	1
DAI0	sclk0	Single Flop	2	5
	sclk1	Single Flop	2	
	sysclk	Not Required	1	
DAI1	sclk0	Single Flop	2	6
	sclk1	Single Flop	2	
	sysclk	Not Required	1	
MLB0	sysclk	Not Required	1	9
EMDMA0	sysclk	Not Required	1	13
EMDMA1	sysclk	Not Required	1	14

Table 4-2: DPM0_PER_DIS0 Register Mapping (Continued)

Peripheral Name	Gated Module Clocks	Type of sync on PER-DIS bit	Effect latency (In cycles of module clocks)	DPM0_PER_DISn bit
CRYPTO ACCELERATOR-0 (EIP-150/PKP)	sysclk	Not Required	1	16
CRYPTO ACCELERATOR-1 (EIP-93/SPE)	sysclk	Not Required	1	17
SMPU-SPIF	sysclk	Not Required	1	19
SMPU-L2CTL-CL2-0	sysclk	Not Required	1	20
SMPU-L2CTL-DL2-0	sysclk	Not Required	1	21
SMPU-DMC0	sysclk	Not Required	1	27

Table 4-3: DPM0_PER_DIS1 Register Mapping

Peripheral Name	Gated Module Clocks	Type of sync on PER-DIS bit	Effect latency (In cycles of module clocks)	DPM0_PER_DISn bit
TWI0	sclk0	Single Flop	2	0
TWI1	sclk0	Single Flop	2	1
TWI2	sclk0	Single Flop	2	2
TWI3	sclk0	Single Flop	2	3
TWI4	sclk0	Single Flop	2	4
TWI5	sclk0	Single Flop	2	5
OSPI	sysclk	Not Required	1	27

ADSP-2156x DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 4-4: ADSP-2156x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_PER_DIS0	Peripherals Disable Register 0
DPM_PER_DIS1	Peripherals Disable Register 1
DPM_REVID	Revision ID
DPM_STAT	Status Register

Control Register

The `DPM_CTL` register controls sleep modes selections and PLL operations of the DPM. A write protect feature permits locking out changes to this register.

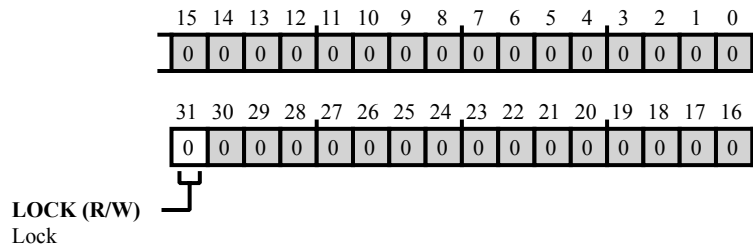


Figure 4-1: DPM_CTL Register Diagram

Table 4-5: DPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>DPM_CTL.LOCK</code> bit is set, the <code>DPM_CTL</code> register is read only (locked).
		0 Unlock
		1 Lock

Peripherals Disable Register 0

The `DPM_PER_DIS0` register is used to shut off the clocks to peripherals that are not used in an application in order to save clock switching power.

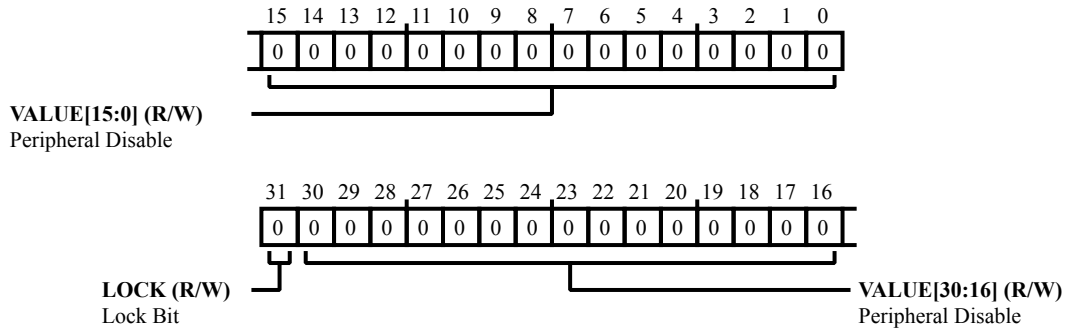


Figure 4-2: DPM_PER_DIS0 Register Diagram

Table 4-6: DPM_PER_DIS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit.
30:0 (R/W)	VALUE	Peripheral Disable. The <code>DPM_PER_DIS0.VALUE</code> bits are used to shut of clocks in individual peripherals.
	0	FIR0
	1	IIR0
	2	Reserved
	3	Reserved
	4	Reserved
	5	DAI0
	6	DAI1
	7	Reserved
	8	Reserved
	9	MLB0
	10	Reserved
	11	Reserved
	12	Reserved
	13	EMDMA0

Table 4-6: DPM_PER_DIS0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		14 EMDMA1
		15 Reserved
		16 CRYPTO ACCELERATOR- 0 (EIP-150/PKP)
		17 CRYPTO ACCELERATOR- 1 (EIP-93/SPE)
		18 Reserved
		19 SMPU-SPIF(SPI2)
		20 SMPU-L2CTL-CL2-0
		21 SMPU-L2CTL-DL2-0
		22 Reserved
		23 Reserved
		24 Reserved
		25 Reserved
		26 Reserved
		27 SMPU-DMC0
		28 Reserved
		29 Reserved
		30 Reserved

Peripherals Disable Register 1

The `DPM_PER_DIS1` register is used to shut off the clocks to peripherals that are not used in an application to save clock switching power.

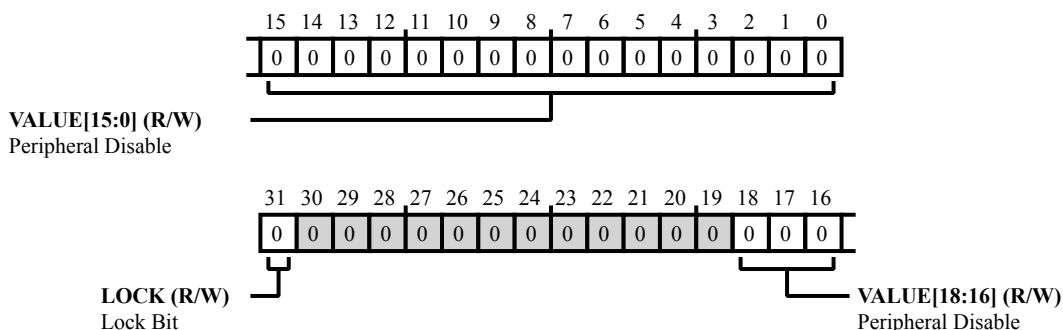


Figure 4-3: DPM_PER_DIS1 Register Diagram

Table 4-7: DPM_PER_DIS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit.
18:0 (R/W)	VALUE	Peripheral Disable. The <code>DPM_PER_DIS1.VALUE</code> bits are used to shut of clocks in individual peripherals.
		0 TWI0
		1 TWI1
		2 TWI2
		3 TWI3
		4 TWI4
		5 TWI5
		6 OSPI
		7 Reserved
		8 Reserved
		9 Reserved
		10 Reserved
		11 Reserved
		12 Reserved
		13 Reserved

Table 4-7: DPM_PER_DIS1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		14	Reserved
		15	Reserved
		16	Reserved
		17	Reserved
		18	Reserved

Revision ID

The `DPM_REVID` register provides the revision of the DPM module.

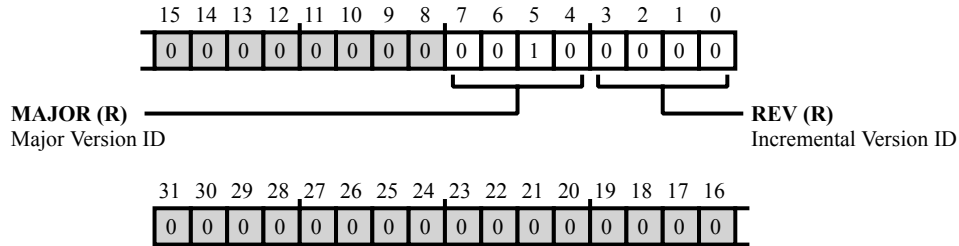


Figure 4-4: DPM_REVID Register Diagram

Table 4-8: DPM_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Status Register

The `DPM_STAT` register contains bits that report the state of the module and various errors.

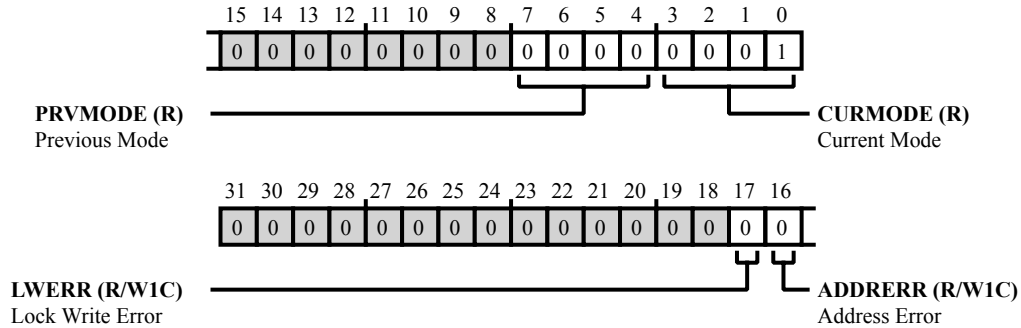


Figure 4-5: DPM_STAT Register Diagram

Table 4-9: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W1C)	LWERR	Lock Write Error. The <code>DPM_STAT.LWERR</code> bit indicates that a write transaction attempted an access to a write protected register. Triggers the <code>DPMLV_PSLVERR</code> interrupt.
		0 Inactive
		1 Active
16 (R/W1C)	ADDRERR	Address Error. The <code>DPM_STAT.ADDRERR</code> bit indicates that a read or write transaction attempted an access to an unimplemented address or a write transaction attempted an access to a read only register or accesses are non aligned. Triggers the <code>DPMLV_PSLVERR</code> interrupt.
		0 Inactive
		1 Active
7:4 (R/NW)	PRVMODE	Previous Mode. The <code>DPM_STAT.PRVMODE</code> bit field indicates the previous mode of the the module.
		0 Reset
		1 Full-On
		2 Reserved
		3 Reserved
		4 Reserved
5-15 Reserved		

Table 4-9: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/NW)	CURMODE	Current Mode. The DPM_STAT.CURMODE bit field indicates the current mode of the the module.	
		0	Reserved
		1	Full-On
		2	Reserved
		3	Reserved
		4-15	Reserved

5 Reset Control Unit (RCU)

The Reset Control Unit (RCU) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. While core resets and software system resets are controlled directly in the RCU, hardware resets can come from the TRU, SEC, or CGU Oscillator Watchdog.

RCU Features

The RCU module supports the following features:

- Hardware reset through the `SYS_HWRST` pin
- Software system reset through the RCU control (`RCU_CTL`) register
- Hardware system reset through:
 - TRU module
 - SEC module (System Fault Unit)
- A clock not good reset state (safe state of chip under reset) from the Oscillator Watchdog.
- Core reset through RCU Core Reset Output (`RCU_CRCTL`) register

RCU Functional Description

This section provides information on the function of RCU module.

NOTE: In 2156x series of processors, there is only one core. The terminology *n* when describing number of cores (for example, core[n]) must be considered always as 0 (core 0).

Hardware reset using `SYS_HWRST` pin

Asserting the `SYS_HWRST` pin resets all functional units.

Hardware reset through RCU

The RCU can perform a full system reset which can be initiated through hardware blocks like the SEC, the TRU, and the oscillator watchdog.

Software reset using RCU registers

Setting the `RCU_STAT.SWRST` bit issues a software reset for all system units except the `RCU_BCODE`, `RCU_CRCTL` and `RCU_STAT` registers.

Core reset RCU registers

A core can be individually reset by software, or by setting the `RCU_CRCTL.CR[n]` bit.

ADSP-2156x RCU Register List

The Reset Control Unit (RCU) controls how all the functional units in the processor enter and exit Reset. Differences in functional requirements and clocking constraints exist (units in different clock domains have to enter reset asynchronously, but units exit reset in a deterministic way), and these differences define how reset signals are generated. Reset signals propagate through all functional units asynchronously. For more information on RCU functionality, see the RCU register descriptions.

Table 5-1: ADSP-2156x RCU Register List

Name	Description
<code>RCU_BCODE</code>	Boot Code Register
<code>RCU_CRCTL</code>	Core Reset Outputs Control Register
<code>RCU_CRSTAT</code>	Core Reset Outputs Status Register
<code>RCU_CTL</code>	Control Register
<code>RCU_MSG</code>	Message Register
<code>RCU_MSG_CLR</code>	Message Clear Bits Register
<code>RCU_MSG_SET</code>	Message Set Bits Register
<code>RCU_SIDIS</code>	System Interface Disable Register
<code>RCU_SISTAT</code>	System Interface Status Register
<code>RCU_SRRQSTAT</code>	System Reset Request Status Register
<code>RCU_STAT</code>	Status Register
<code>RCU_SVECT0</code>	Software Vector Register 0
<code>RCU_SVECT_LCK</code>	SVECT Lock Register

ADSP-2156x RCU Trigger List

Table 5-2: ADSP-2156x RCU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
		None	

Table 5-3: ADSP-2156x RCU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
44	RCU0_SYSRST0	RCU0 System Reset 0	Pulse
45	RCU0_SYSRST1	RCU0 System Reset 1	Pulse

RCU Definitions

To make the best use of the RCU, it is useful to understand the terms in this section.

The target or source defines the following are types of resets.

Hardware Reset (by target)

All functional units except a small subsection of debug interfaces are set to their default states. State is lost in all non-volatile storage.

System Reset (by target)

All functional units except the RCU, flash interface, and debug are set to their default states.

Core n Only Reset (by target)

Affects Core n only. The system software must guarantee that a bus master cannot access the core in reset state.

Hardware Reset (by source)

The $\overline{\text{SYS_HWRST}}$ input signal is asserted active (pulled low).

System Reset (by source)

Software can trigger the reset by writing to the [RCU_CTL](#) register or by another functional unit such as the TRU or any of the generic reset inputs.

RCU Architectural Concepts

To understand the architecture of the RCU, it is important to consider the reset sources and how differing resets affect the functional units of the processor.

The RCU provides the hardware that controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. For example, units in different clock domains must enter reset asynchronously but exit reset in a deterministic way.

The program must guarantee that none of the reset functions put the system in an undefined state or cause resources to stall. The program must guarantee that there is no pending system activity involving Core *n* before it is reset.

The *RCU Reset Sources* table defines how reset sources affect the different functional units.

Table 5-4: RCU Reset Sources

Reset Source	Reset Type	Affected Functional Units
<code>SYS_HWRST</code> pin assertion	Hardware Reset	All functional units, except RTC (if present)
<code>SYS_HWRST</code> pin assertion	Hardware Reset	All functional units
SYSCLK clock domain system reset by the Fault Unit (FMU) or the Trigger Routing Unit (TRU)	System Reset	All functional units, except: <ul style="list-style-type: none"> • RTC (if present) • RCU_STAT • RCU_BCODE • the units on the VDDEXT power domain
RCU_CTL.SYSRST bit set (software triggered reset)	System Reset	All functional units, except: <ul style="list-style-type: none"> • RTC (if present) • RCU_STAT • RCU_BCODE • the units on the VDDEXT power domain
RCU_CRCTL.CR[n] bit set (software triggered reset)	Core Only Reset	Core <i>n</i> only

RCU Status and Error Signals

The `RCU_STAT` register reflects status and error information. There are three kinds of errors that can occur in the RCU. The reset out error is triggered when `RSTOUT` is both asserted and deasserted at the same time. The lock write error occurs if an attempt is made to write a lock RCU register. The address error occurs if a read-only register is written to or if an attempt is made to a reserved address within the RCU MMR address range.

ADSP-2156x RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 5-5: ADSP-2156x RCU Register List

Name	Description
<code>RCU_BCODE</code>	Boot Code Register

Table 5-5: ADSP-2156x RCU Register List (Continued)

Name	Description
RCU_CRCTL	Core Reset Outputs Control Register
RCU_CRSTAT	Core Reset Outputs Status Register
RCU_CTL	Control Register
RCU_MSG	Message Register
RCU_MSG_CLR	Message Clear Bits Register
RCU_MSG_SET	Message Set Bits Register
RCU_SIDIS	System Interface Disable Register
RCU_SISTAT	System Interface Status Register
RCU_SRRQSTAT	System Reset Request Status Register
RCU_STAT	Status Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT_LCK	SVECT Lock Register

Boot Code Register

The `RCU_BCODE` register can be used to determine if and how core boots. This register is set to its default values by RESET.

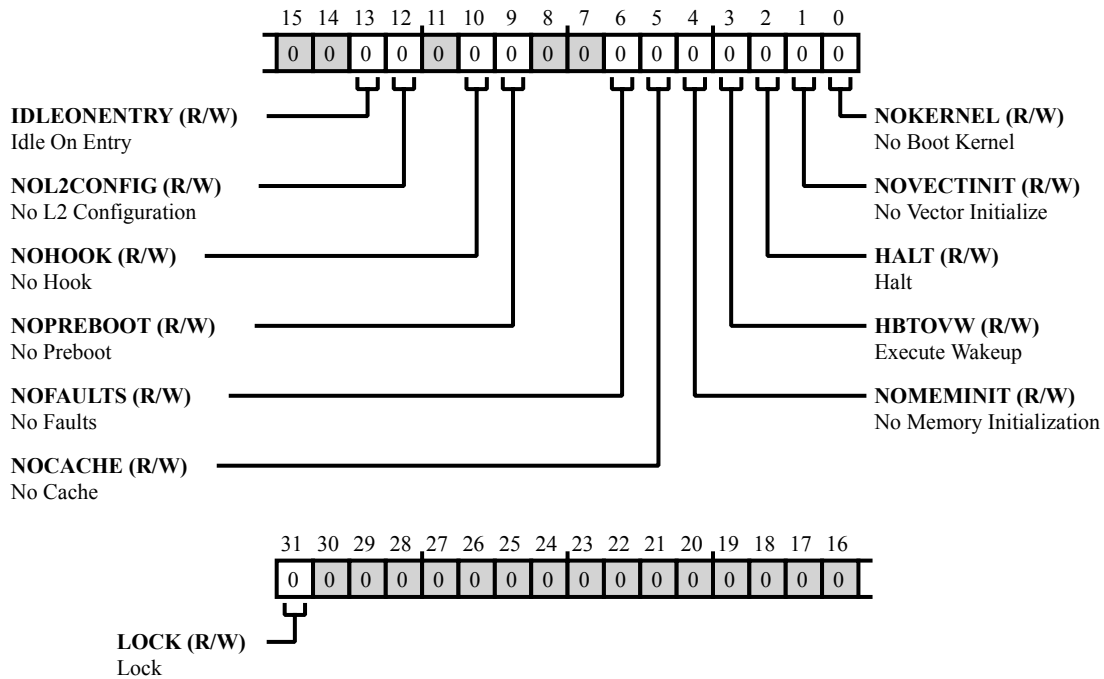


Figure 5-1: RCU_BCODE Register Diagram

Table 5-6: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>RCU_BCODE.LOCK</code> bit is set, the <code>RCU_BCODE</code> register is read only (locked).
		0 Unlock
		1 Lock
13 (R/W)	IDLEONENTRY	Idle On Entry. The <code>RCU_BCODE.IDLEONENTRY</code> bit configures the RCU to enter the idle state at startup.
		0 Do not enter idle state
		1 Enter idle state

Table 5-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	NOL2CONFIG	No L2 Configuration. The RCU_BCODE.NOL2CONFIG bit configures the RCU to not perform the L2 memory configuration.
		0 Configure L2 memory
		1 Do not configure L2 memory
10 (R/W)	NOHOOK	No Hook. The RCU_BCODE.NOHOOK bit configures the RCU to not perform the hook routine.
		0 Perform hook routine
		1 Do not perform hook routine
9 (R/W)	NOPREBOOT	No Preboot. The RCU_BCODE.NOPREBOOT bit configures the RCU to not perform the customer preboot routine.
		0 Perform preboot
		1 Do not perform preboot
6 (R/W)	NOFAULTS	No Faults. The RCU_BCODE.NOFAULTS bit configures the RCU to not perform fault initialization.
		0 Perform fault initialization
		1 Do not perform fault initialization
5 (R/W)	NOCACHE	No Cache. The RCU_BCODE.NOCACHE bit configures the RCU to not perform a cache initialization and to not enable the cache.
		0 Enable and initialize cache
		1 Do not initialize or enable cache
4 (R/W)	NOMEMINIT	No Memory Initialization. The RCU_BCODE.NOMEMINIT bit configures the RCU to not perform a memory initialization.
		0 Perform memory initialization
		1 Do not perform memory initialization

Table 5-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	HBTOVW	Execute Wakeup. The RCU_BCODE.HBTOVW bit configures the RCU to execute a wakeup.
		0 Do not wakeup
		1 Execute wakeup
2 (R/W)	HALT	Halt. The RCU_BCODE.HALT bit configures the RCU to execute the no boot routine.
		0 Do not execute routine
		1 Execute routine
1 (R/W)	NOVECTINIT	No Vector Initialize. The RCU_BCODE.NOVECTINIT bit configures the RCU to not vector to the application.
		0 Vector
		1 Do not vector
0 (R/W)	NOKERNEL	No Boot Kernel. The RCU_BCODE.NOKERNEL bit configures the RCU to not execute the boot kernel.
		0 Execute boot kernel
		1 Do not execute boot kernel

Core Reset Outputs Control Register

The RCU core reset control n registers (`RCU_CRCTL`) include a lock bit (`RCU_CRCTL.LOCK`) and a core reset bit (`RCU_CRCTL.CR[n]`) for each core reset signal on the product.

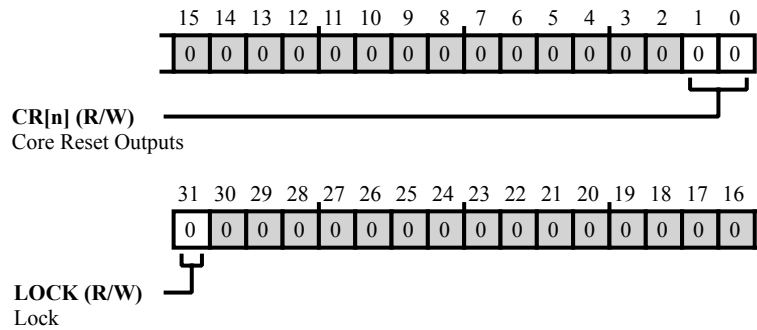


Figure 5-2: RCU_CRCTL Register Diagram

Table 5-7: RCU_CRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>RCU_CRCTL.LOCK</code> bit is set, the <code>RCU_CRCTL</code> register is read only (locked).
		0 Unlock
		1 Lock
1:0 (R/W)	CR[n]	Core Reset Outputs.
		The <code>RCU_CRCTL.CR[n]</code> bits control <code>CRES[1:0]</code> core reset signals. The <code>RCU_CRES[n]</code> signals can be individually controlled. They are reset to their default value by a hard reset or a system reset. For each <code>RCU_CRES[n]</code> , the selected <code>RCU0_CRMSKi[n]</code> bit is cleared.
		0 RCU_CRES[1:0] Deasserted
		3 RCU_CRES[1:0] Asserted

Core Reset Outputs Status Register

The RCU core reset status register (`RCU_CRSTAT`) contains status bits, indicating which core reset signals have been asserted.

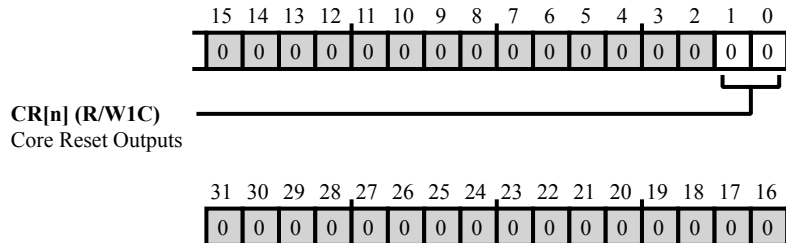


Figure 5-3: RCU_CRSTAT Register Diagram

Table 5-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W1C)	CR[n]	<p>Core Reset Outputs.</p> <p>The <code>RCU_CRSTAT.CR[n]</code> bits indicate which cores have been reset since the last time the bit was cleared. Bits masked by <code>CORE_DISABLE_MASK[15:0]</code> are permanently disabled and the corresponding CR bits set. CR bits are sticky, they need to be cleared by software.</p>
		0 RCU_CRES[1:0] deasserted. CR[n] corresponds to RCU_CRES[n].
		3 RCU_CRES[1:0] were asserted since the last time bits were cleared. CR[n] corresponds to RCU_CRES[n].

Control Register

The RCU control register (`RCU_CTL`) provides a register lock, enables for the core and system reset requests inputs and control for the Reset Output pin.

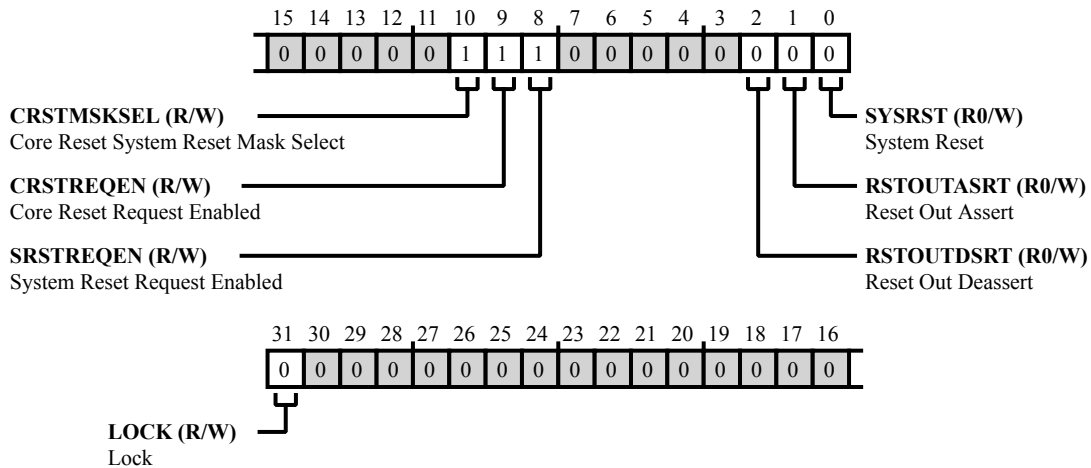


Figure 5-4: RCU_CTL Register Diagram

Table 5-9: RCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		0 Unlock
		1 Lock
10 (R/W)	CRSTMSKSEL	Core Reset System Reset Mask Select. The <code>RCU_CTL.CRSTMSKSEL</code> bit selects the core reset system reset mask. This bit is cleared by a hard reset.
9 (R/W)	CRSTREQEN	Core Reset Request Enabled.
		The <code>RCU_CTL.CRSTREQEN</code> bit controls whether the SYSCLK domain source(s) of reset is/are enabled to reset the core(s) when asserted. This bit is cleared by hard reset or any system reset event.
		0 Disabled
		1 Enabled

Table 5-9: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	SRSTREQEN	System Reset Request Enabled. The RCU_CTL.SRSTREQEN bit controls whether the SYSCLK domain sources of reset are enabled to do a system reset when asserted. This bit is cleared by a hard reset.
		0 Disabled
		1 Enabled
2 (R0/W)	RSTOUTDSRT	Reset Out Deassert. The RCU_CTL.RSTOUTDSRT bit controls the deassertion of the system reset pin.
		0 No Action
		1 Deassert RSTOUT
1 (R0/W)	RSTOUTASRT	Reset Out Assert. The RCU_CTL.RSTOUTASRT bit controls assertion of the system reset pin.
		0 No Action
		1 Assert RSTOUT
0 (R0/W)	SYSRST	System Reset. The RCU_CTL.SYSRST bit provides reset for all system units.
		0 No Action
		1 System Reset

Message Register

The `RCU_MSG` is a general-purpose register. It is intended to provide flexibility for Boot ROM code and to pass predefined variables to the debugger. Please see the Booting chapter for product-specific details.

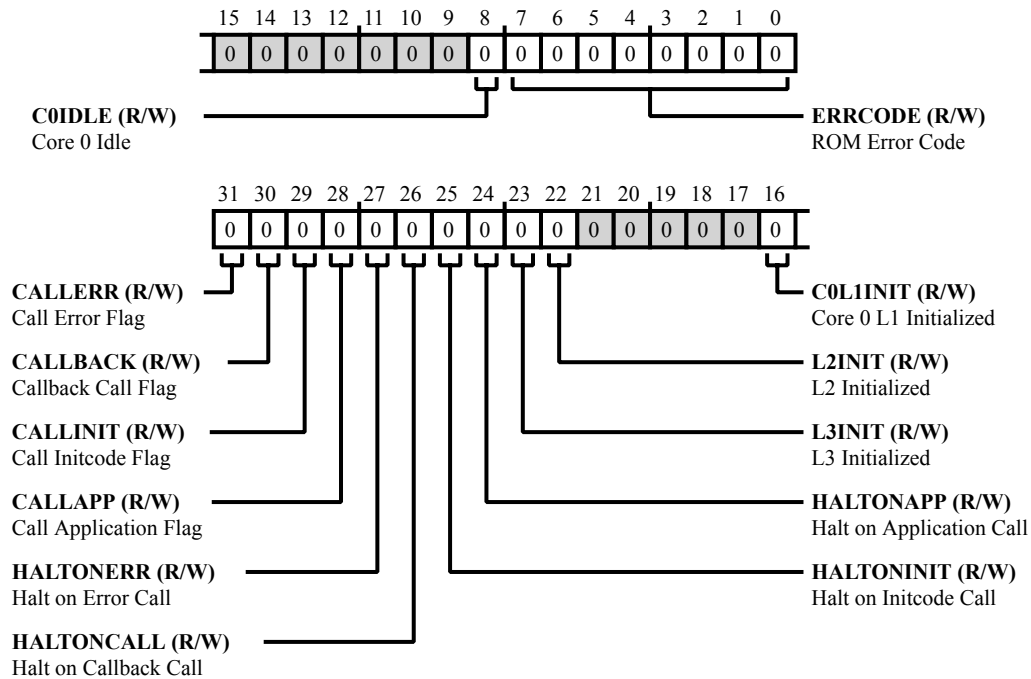


Figure 5-5: RCU_MSG Register Diagram

Table 5-10: RCU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CALLERR	Call Error Flag. The <code>RCU_MSG.CALLERR</code> bit indicates that a flag has been set by the boot code prior to an error call.
		0 Flag not set
		1 Flag set
30 (R/W)	CALLBACK	Callback Call Flag. The <code>RCU_MSG.CALLBACK</code> bit indicates that a flag has been set by the boot code prior to a callback call.
		0 Flag not set
		1 Flag set

Table 5-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	CALLINIT	Call Initcode Flag. The RCU_MSG.CALLINIT bit indicates that a flag has been set by the boot code prior to an initcode call.
		0 Flag not set
		1 Flag set
28 (R/W)	CALLAPP	Call Application Flag. The RCU_MSG.CALLAPP bit indicates that a flag has been set by the boot code prior to an application call.
		0 Flag not set
		1 Flag set
27 (R/W)	HALTONERR	Halt on Error Call. The RCU_MSG.HALTONERR bit generates an emulation exception prior to an error call.
		0 Do not generate exception
		1 Generate exception
26 (R/W)	HALTONCALL	Halt on Callback Call. The RCU_MSG.HALTONCALL bit generates an emulation exception prior to a callback call.
		0 Do not generate exception
		1 Generate exception
25 (R/W)	HALTONINIT	Halt on Initcode Call. The RCU_MSG.HALTONINIT bit generates an emulation exception prior to an initcode call.
		0 Do not generate exception
		1 Generate exception
24 (R/W)	HALTONAPP	Halt on Application Call. The RCU_MSG.HALTONAPP bit generates an emulation exception prior to an application call.
		0 Do not generate exception
		1 Generate exception

Table 5-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	L3INIT	L3 Initialized. The RCU_MSG.L3INIT bit indicates that the L3 resource is initialized.
		0 Resource not initialized
		1 Resource initialized
22 (R/W)	L2INIT	L2 Initialized. The RCU_MSG.L2INIT bit indicates that the L2 resource is initialized.
		0 Resource not initialized
		1 Resource initialized
16 (R/W)	C0L1INIT	Core 0 L1 Initialized. The RCU_MSG.C0L1INIT bit indicates that the core 0 L1 resource is initialized.
		0 Resource not initialized
		1 Resource initialized
8 (R/W)	C0IDLE	Core 0 Idle. The RCU_MSG.C0IDLE bit indicates that core 0 is in a safe idle state in ROM.
7:0 (R/W)	ERRCODE	ROM Error Code. The RCU_MSG.ERRCODE bit indicates the error code of the ROM. It is valid only when in the error handler.

Message Clear Bits Register

The `RCU_MSG_CLR` register is used to clear bits in `RCU_MSG` register. Reading this register returns `0x00000000`.

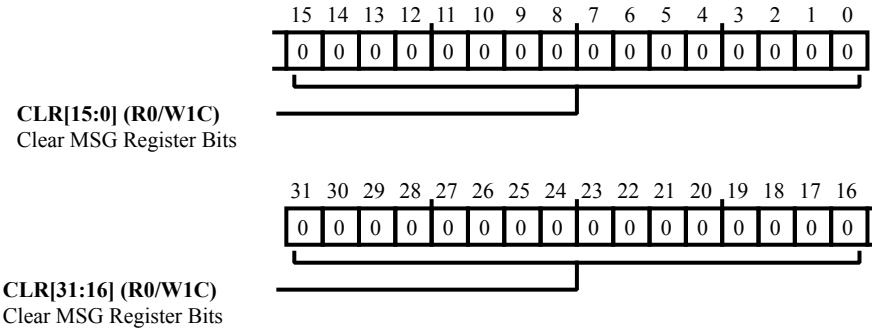


Figure 5-6: `RCU_MSG_CLR` Register Diagram

Table 5-11: `RCU_MSG_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1C)	CLR	Clear MSG Register Bits. The <code>RCU_MSG_CLR</code> . CLR bit resets MSG bit n.

Message Set Bits Register

The `RCU_MSG_SET` register is used to set bits in `RCU_MSG` register. Reading this register returns 0x00000000.

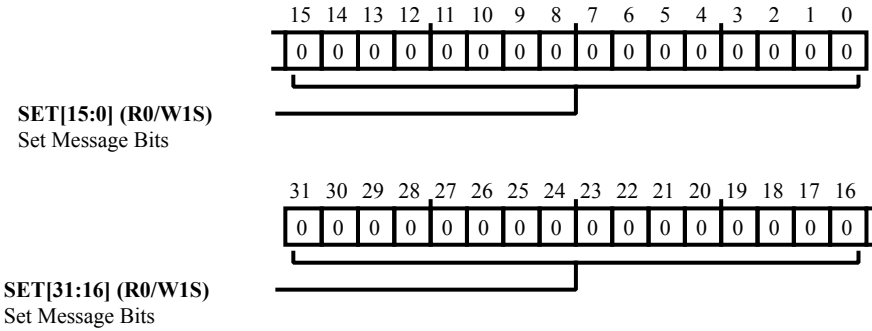


Figure 5-7: RCU_MSG_SET Register Diagram

Table 5-12: RCU_MSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1S)	SET	Set Message Bits. The <code>RCU_MSG_SET.SET</code> bit sets <code>MSG</code> bit n.

System Interface Disable Register

The RCU system interface disable register (`RCU_SIDIS`) lets the RCU assert a system interface disable request to functional units in the processor. This register is set to its default values by a hard reset or any system reset event. For information on mapping between `RCU_SIDIS` bits and functional units, see the RCU functional description.

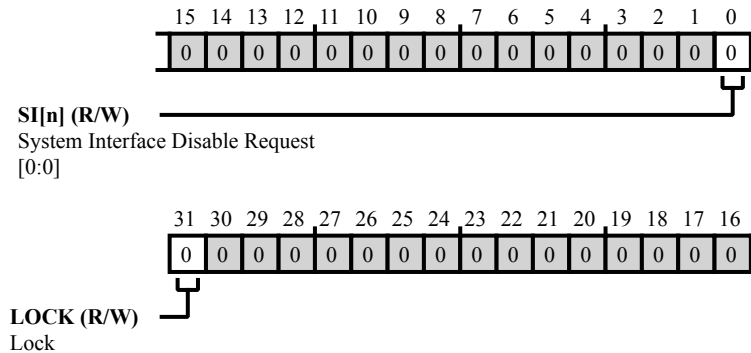


Figure 5-8: RCU_SIDIS Register Diagram

Table 5-13: RCU_SIDIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>RCU_SIDIS.LOCK</code> bit is set, the <code>RCU_SIDIS</code> register is read only (locked).
		0 Unlock
		1 Lock
0 (R/W)	SI[n]	System Interface Disable Request [0:0].
		Each <code>RCU_SIDIS.SI[n]</code> bit corresponds to a functional unit in the processor that supports the system interface disable request-acknowledge protocol.
		0 <code>RCU_SI_DISABLE_REQ[0:0]</code> deasserted
		1 <code>RCU_SI_DISABLE_REQ[0:0]</code> asserted

System Interface Status Register

The RCU system interface status register (`RCU_SISTAT`) indicates whether a functional unit has or has not acknowledged an RCU unit disable request.

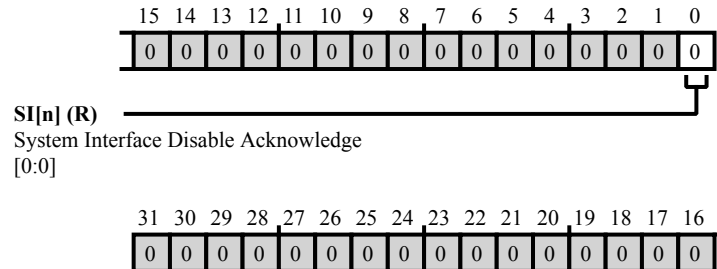


Figure 5-9: RCU_SISTAT Register Diagram

Table 5-14: RCU_SISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	SI[n]	System Interface Disable Acknowledge [0:0]. The <code>RCU_SISTAT.SI[n]</code> bit indicates whether a functional unit has or has not acknowledged an RCU unit disable request.
		0 No Acknowledge
		1 SI_DISABLE_ACK[0:0] asserted

System Reset Request Status Register

The RCU system reset request status register ([RCU_SRRQSTAT](#)) contains status bits, indicating which system reset request input triggered a system reset. This register is set to its default values by a hard reset.

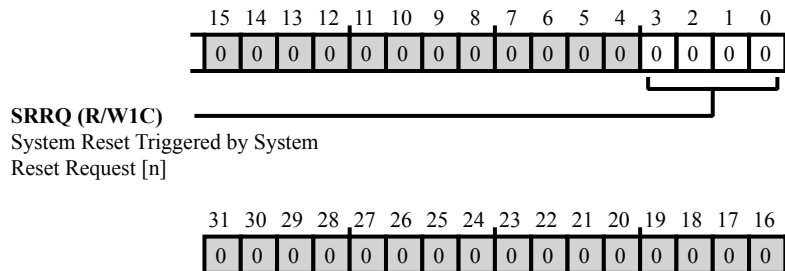


Figure 5-10: RCU_SRRQSTAT Register Diagram

Table 5-15: RCU_SRRQSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1C)	SRRQ	System Reset Triggered by System Reset Request [n]. The <code>RCU_SRRQSTAT.SRRQ</code> bits are set by the assertion of the corresponding system reset request input and deasserted by writing "1" to the bit. The RCU_SRRQSTAT register is cleared by a hard reset.

Status Register

The RCU status register (`RCU_STAT`) contains status bits for all RCU reset sources, reset status, and boot mode inputs. Status bits for reset sources are sticky and can be cleared by software. Error status bits are cleared by any reset event.

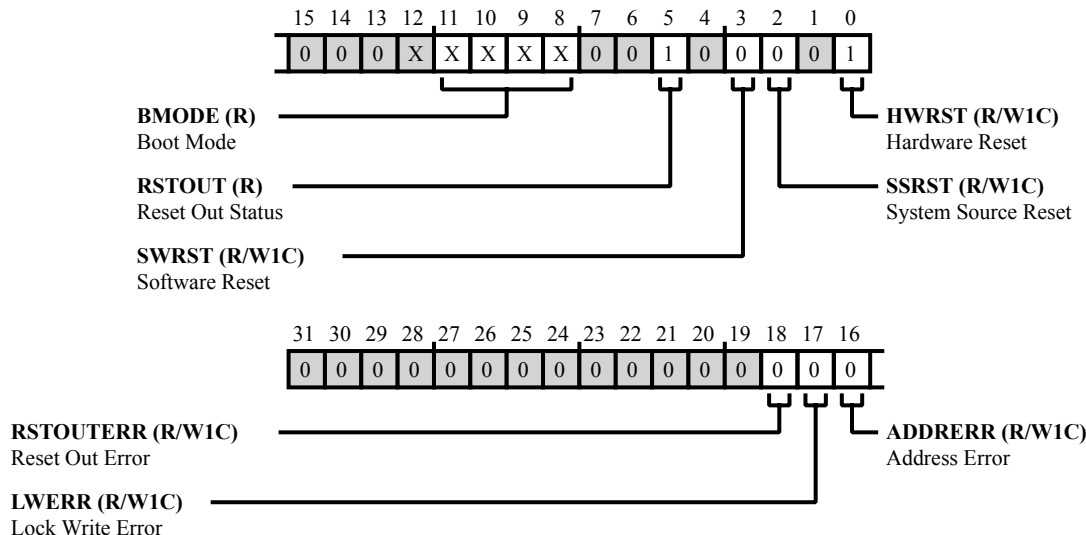


Figure 5-11: RCU_STAT Register Diagram

Table 5-16: RCU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W1C)	RSTOUTERR	Reset Out Error. The <code>RCU_STAT.RSTOUTERR</code> bit indicates (if set) that a write attempted to set the <code>RCU_CTL.RSTOUTASRT</code> and <code>RCU_CTL.RSTOUTDSRT</code> simultaneously. This condition triggers a bus error.
		0 No Error
		1 Error Occurred
17 (R/W1C)	LWERR	Lock Write Error. The <code>RCU_STAT.LWERR</code> bit indicates (when set) there was an attempted write to an RCU register while the <code>RCU_CTL.LOCK</code> bit was set and the global lock bit is enabled (<code>SPU_CTL.GLCK</code> bit = 1). This status bit is sticky; write-1-to-clear
		0 No Error
		1 Error Occurred

Table 5-16: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W1C)	ADDRERR	Address Error. The RCU_STAT.ADDRERR bit indicates that the RCU generated an address error. This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
11:8 (R/NW)	BMODE	Boot Mode. The RCU_STAT.BMODE bits indicate the input on the boot mode pins.
5 (R/NW)	RSTOUT	Reset Out Status. The RCU_STAT.RSTOUT bit indicates the assertion status of the system reset pin.
		0 RSTOUT Deasserted
		1 RSTOUT Asserted
3 (R/W1C)	SWRST	Software Reset. The RCU_STAT.SWRST bit indicates that a system reset (which was triggered by software) has occurred since the last time a hardware reset occurred or since the RCU_STAT.SWRST bit was cleared by software.
		0 Inactive
		1 Reset Occurred
2 (R/W1C)	SSRST	System Source Reset. The RCU_STAT.SSRST bit indicates that a system reset triggered by hardware in the system clock domain, clock A domain, or clock B domain has occurred since the last time a hardware reset occurred or since the RCU_STAT.SSRST bit was cleared by software.
		0 Inactive
		1 Reset Occurred
0 (R/W1C)	HWRST	Hardware Reset. The RCU_STAT.HWRST bit indicates that a hardware reset has occurred.
		0 Inactive
		1 Reset Occurred

Software Vector Register 0

The `RCU_SVECT0` register contains the default location of the first instruction to execute after a reset.

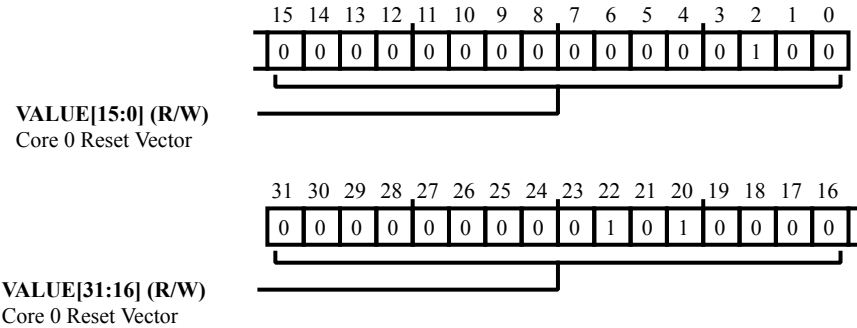


Figure 5-12: RCU_SVECT0 Register Diagram

Table 5-17: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Core 0 Reset Vector. The <code>RCU_SVECT0.VALUE</code> bit field contains the default location of the first instruction to execute after a reset.

SVECT Lock Register

The RCU software vector lock register (`RCU_SVECT_LCK`) provides a register lock and software vector n enable bits for each processor core on the product. This register is set to its default values by a hard reset or any system reset event.

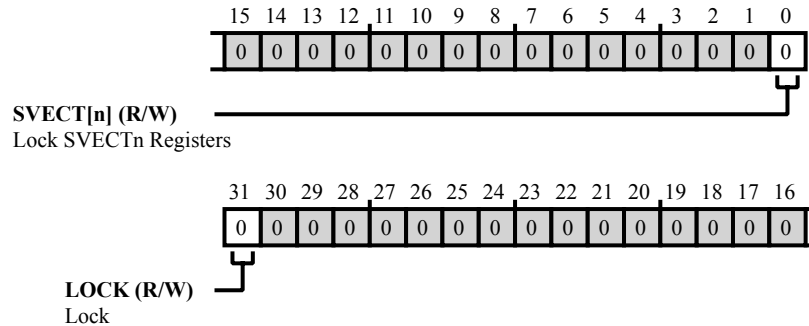


Figure 5-13: RCU_SVECT_LCK Register Diagram

Table 5-18: RCU_SVECT_LCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>RCU_SVECT_LCK.LOCK</code> bit is set, the <code>RCU_SVECT_LCK</code> register is read only (locked).
		0 Unlock
		1 Lock
0 (R/W)	SVECT[n]	Lock SVECTn Registers. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>RCU_SVECT_LCK.SVECT[n]</code> bit is set, the SVECT registers are read only (locked).

6 System Event Controller (SEC)

System event management is the responsibility of the system event controller (SEC). The SEC manages the configuration of all system event sources. The SEC also manages the propagation of system events to all connected cores and the system fault interface.

All of the peripheral interrupts are routed using a single SEC interrupt to the desired core. The SEC allows programmability of the peripheral interrupt's priority, supporting up to 256 priority levels that are arbitrated within the SEC itself. The SEC also allows these interrupts to be grouped and masked by priority level and provides the flexibility to choose which core(s) the interrupt is routed to.

The SEC also supports self-nesting of interrupts, which is required when sharing a single interrupt request to an individual core, as this allows for a higher-priority peripheral interrupt to be passed to the core while it is currently servicing a lower-priority peripheral interrupt. For more information, refer to “Self-Nesting Mode for System Event Controller Interrupt (SECI)” in the *SHARC+ Core Programming Reference*.

SEC Features

The following list describes the system event controller features.

- Comprehensive system event source management including interrupt enable, fault enable, priority, core mapping, and source grouping.
- Fault management including fault action configuration, timeout, external indication, and system reset.
- Determinism where all system events have the same propagation delay and provide unique identification of a specific system event source.
- Distributed programming model where each system event source control and all status fields are independent of all others.
- Slave control port which provides access to all SEC registers for configuration, status, and interrupt or fault service model.
- Global locking supports a register level protection model to prevent writes to “locked” registers.

SEC Functional Description

The following sections provide a functional description of the SEC.

The *SEC/GIC Interrupt Signal Flow* figure shows an overview of the interrupt systems.

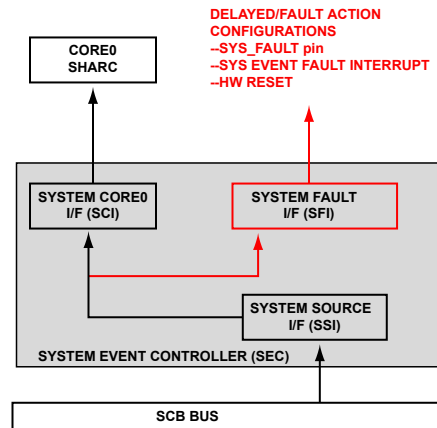


Figure 6-1: SEC Interrupt Signal Flow

ADSP-2156x SEC Register List

The System Event Controller (SEC) manages the system fault sources, including control features such as enable/disable, priority, and active/pending source status. For more information on SEC functionality, see the SEC register descriptions.

Table 6-1: ADSP-2156x SEC Register List

Name	Description
SEC_CACT[n]	SCI Active Register n
SEC_CCTL[n]	SCI Control Register n
SEC_CGMSK[n]	SCI Group Mask Register n
SEC_CPLVL[n]	SCI Priority Level Register n
SEC_CPMSK[n]	SCI Priority Mask Register n
SEC_CPND[n]	Core Pending Register n
SEC_CSID[n]	SCI Source ID Register n
SEC_CSTAT[n]	SCI Status Register n
SEC_END	Global End Register
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register

Table 6-1: ADSP-2156x SEC Register List (Continued)

Name	Description
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

ADSP-2156x SEC Interrupt List

Table 6-2: ADSP-2156x SEC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	SEC0_ERR	SEC0 Error	Level	

ADSP-2156x SEC Trigger List

Table 6-3: ADSP-2156x SEC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
66	SEC0_FAULT	SEC0 Fault	Edge

Table 6-4: ADSP-2156x SEC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-2156x Interrupt List

Table 6-5: ADSP-2156x Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	SEC0_ERR	SEC0 Error	Level	
1	CGU0_EVT	CGU0 Event	Edge	
2	CGU1_EVT	CGU1 Event	Edge	
3	WDOG0_EXP	WDOG0 Expiration	Level	
4	WDOG1_EXP	WDOG1 Expiration	Level	
5	OTPC0_ERR	OTPC0 Dual-bit error	Level	
6	TMU0_FAULT	TMU0 Fault		
7	TMU0_ALERT	TMU0 Fault		
8	TAPC_KEYFAIL	TAPC Test/User Key Fail Interrupt	Edge	
9	L2CTL0_ECC_ERR	L2CTL0 ECC Error	Level	
10	Reserved	Reserved	Reserved	Reserved
11	L2CTL0_EVT	L2CTL0 Scrub/Initialization Done	Level	
12	MEC0_EEIRQ0	MEC0 ECC Error Interrupt Request		
13	MEC0_EWIRQ0	MEC0 ECC Warning Interrupt Request		
14	MEC0_PEIRQ0	MEC0 Parity Error Interrupt Request	Level	
15	MEC0_PEIRQ1	MEC0 Parity Error Interrupt Request	Level	
16	C0_DATA_READ_ERR	Core 0 Data Read Interrupt		
17	C0_DATA_WRITE_ERR	Core 0 Data Write Interrupt		
18	C0_INST_READ_ERR	Core 0 Instruction Read Interrupt		
19	C0_IDLE	Core 0 Idle Interrupt		
20	DAI0_IRQH	DAI0 High priority	None	
21	DAI1_IRQH	DAI1 High priority	None	
22	TIMER0_TMR00	TIMER0 Timer 0	Level	
23	TIMER0_TMR01	TIMER0 Timer 1	Level	
24	TIMER0_TMR02	TIMER0 Timer 2	Level	
25	TIMER0_TMR03	TIMER0 Timer 3	Level	
26	PINT0_BLOCK	PINT0 Pin Interrupt Block 0	Level	
27	PINT1_BLOCK	PINT1 Pin Interrupt Block 1	Level	
28	PINT2_BLOCK	PINT2 Pin Interrupt Block 2	Level	
29	SOFT0_INT	Software Interrupt 0		
30	SOFT1_INT	Software Interrupt 1		

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
31	SOFT2_INT	Software Interrupt 2		
32	SOFT3_INT	Software Interrupt 3		
33	SOFT4_INT	Software Interrupt 4		
34	SOFT5_INT	Software Interrupt 5		
35	SOFT6_INT	Software Interrupt 6		
36	SOFT7_INT	Software Interrupt 7		
37	SPORT0_A_DMA	SPORT0 Channel A DMA	Level	0
38	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
39	SPORT0_B_DMA	SPORT0 Channel B DMA	Level	1
40	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
41	SPORT1_A_DMA	SPORT1 Channel A DMA	Level	2
42	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
43	SPORT1_B_DMA	SPORT1 Channel B DMA	Level	3
44	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
45	SPORT2_A_DMA	SPORT2 Channel A DMA	Level	4
46	SPORT2_A_STAT	SPORT2 Channel A Status	Level	
47	SPORT2_B_DMA	SPORT2 Channel B DMA	Level	5
48	SPORT2_B_STAT	SPORT2 Channel B Status	Level	
49	SPORT3_A_DMA	SPORT3 Channel A DMA	Level	6
50	SPORT3_A_STAT	SPORT3 Channel A Status	Level	
51	SPORT3_B_DMA	SPORT3 Channel B DMA	Level	7
52	SPORT3_B_STAT	SPORT3 Channel B Status	Level	
53	SPORT4_A_DMA	SPORT4 Channel A DMA	Level	10
54	SPORT4_A_STAT	SPORT4 Channel A Status	Level	
55	SPORT4_B_DMA	SPORT4 Channel B DMA	Level	11
56	SPORT4_B_STAT	SPORT4 Channel B Status	Level	
57	SPORT5_A_DMA	SPORT5 Channel A DMA	Level	12
58	SPORT5_A_STAT	SPORT5 Channel A Status	Level	
59	SPORT5_B_DMA	SPORT5 Channel B DMA	Level	13
60	SPORT5_B_STAT	SPORT5 Channel B Status	Level	
61	SPORT6_A_DMA	SPORT6 Channel A DMA	Level	14

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
62	SPORT6_A_STAT	SPORT6 Channel A Status	Level	
63	SPORT6_B_DMA	SPORT6 Channel B DMA	Level	15
64	SPORT6_B_STAT	SPORT6 Channel B Status	Level	
65	SPORT7_A_DMA	SPORT7 Channel A DMA	Level	16
66	SPORT7_A_STAT	SPORT7 Channel A Status	Level	
67	SPORT7_B_DMA	SPORT7 Channel B DMA	Level	17
68	SPORT7_B_STAT	SPORT7 Channel B Status	Level	
69	SPI2_TXDMA	SPI2 TX DMA Channel	Level	26
70	SPI2_RXDMA	SPI2 RX DMA Channel	Level	27
71	SPI2_STAT	SPI2 Status	Level	
72	SPI2_ERR	SPI2 Error	Level	
73	OSPI0_INT	OSPI0 Error	Level	
74	TIMER0_TMR04	TIMER0 Timer 4	Level	
75	TIMER0_TMR05	TIMER0 Timer 5	Level	
76	TIMER0_TMR06	TIMER0 Timer 6	Level	
77	TIMER0_TMR07	TIMER0 Timer 7	Level	
78	TIMER0_TMR08	TIMER0 Timer 8	Level	
79	TIMER0_TMR09	TIMER0 Timer 9	Level	
80	TIMER0_STAT	TIMER0 Status	Level	
81	LP0_DMA	LP0 DMA Channel		30
82	LP0_STAT	LP0 Status		
83	LP1_DMA	LP1 DMA Channel		36
84	LP1_STAT	LP1 Status		
85	SPI0_TXDMA	SPI0 TX DMA Channel	Level	22
86	SPI0_RXDMA	SPI0 RX DMA Channel	Level	23
87	SPI0_STAT	SPI0 Status	Level	
88	SPI0_ERR	SPI0 Error	Level	
89	SPI1_TXDMA	SPI1 TX DMA Channel	Level	24
90	SPI1_RXDMA	SPI1 RX DMA Channel	Level	25
91	SPI1_STAT	SPI1 Status	Level	
92	SPI1_ERR	SPI1 Error	Level	

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
93	UART0_TXDMA	UART0 Transmit DMA	Level	20
94	UART0_RXDMA	UART0 Receive DMA	Level	21
95	UART0_STAT	UART0 Status	Level	
96	UART1_TXDMA	UART1 Transmit DMA	Level	34
97	UART1_RXDMA	UART1 Receive DMA	Level	35
98	UART1_STAT	UART1 Status	Level	
99	UART2_TXDMA	UART2 Transmit DMA	Level	37
100	UART2_RXDMA	UART2 Receive DMA	Level	38
101	UART2_STAT	UART2 Status	Level	
102	TWI0_DATA	TWI0 Data Interrupt	Level	
103	TWI1_DATA	TWI1 Data Interrupt	Level	
104	TWI2_DATA	TWI2 Data Interrupt	Level	
105	TWI3_DATA	TWI3 Data Interrupt	Level	
106	TWI4_DATA	TWI4 Data Interrupt	Level	
107	TWI5_DATA	TWI5 Data Interrupt	Level	
108	CNT0_STAT	CNT0 Status	Level	
109	CTI0_EVT0	CTI0 Core 0 CTI Event	Level	
110	PKIC0_IRQ	PKIC0 Interrupt	Level	
111	PKTE0_IRQ	PKTE0 Interrupt	Level	
112	TRU0_SLV0	TRU0 Interrupt 4 -- Core 0	Edge	
113	TRU0_SLV1	TRU0 Interrupt 5 -- Core 0	Edge	
114	TRU0_SLV2	TRU0 Interrupt 6 -- Core 0	Edge	
115	TRU0_SLV3	TRU0 Interrupt 7 -- Core 0	Edge	
116	DAI0_IRQL	DAI0 Low Priority Interrupt	None	
117	DAI1_IRQL	DAI1 Low Priority Interrupt	None	
118	FIR0_DMA	FIR0 DMA	Edge	
119	FIR0_STAT	FIR0 Status	Edge	
120	IIR0_DMA	IIR0 DMA	Edge	
121	IIR0_STAT	IIR0 Status	Edge	
122	HADC0_EVT	HADC0 Event	Edge	
123	MLB0_INT0	MLB0 Interrupt 0		

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
124	MLB0_INT1	MLB0 Interrupt 1		
125	MLB0_STAT	MLB0 Status		
126	MDMA3_SRC	Memory DMA Stream 3 Source Channel		43
127	MDMA3_DST	Memory DMA Stream 3 Destination Channel		44
128	MDMA2_SRC	Memory DMA Stream 2 Source Channel		39
129	MDMA2_DST	Memory DMA Stream 2 Destination Channel		40
130	EMDMA0_DONE	EMDMA0 Transfer Done	Edge	
131	EMDMA1_DONE	EMDMA1 Transfer Done	Edge	
132	MDMA0_SRC	Memory DMA Stream 0 Source Channel		8
133	MDMA0_DST	Memory DMA Stream 0 Destination Channel		9
134	MDMA1_SRC	Memory DMA Stream 1 Source Channel		18
135	MDMA1_DST	Memory DMA Stream 1 Destination Channel		19
136	CRC0_DCNTXP	CRC0 Datacount expiration	Level	
137	CRC1_DCNTXP	CRC1 Datacount expiration	Level	
138	CRC0_ERR	CRC0 Error	Level	
139	CRC1_ERR	CRC1 Error	Level	
140	SPORT0_A_DMA_ERR	SPORT0 Channel A DMA Error	Level	
141	SPORT0_B_DMA_ERR	SPORT0 Channel B DMA Error	Level	
142	SPORT1_A_DMA_ERR	SPORT1 Channel A DMA Error	Level	
143	SPORT1_B_DMA_ERR	SPORT1 Channel B DMA Error	Level	
144	SPORT2_A_DMA_ERR	SPORT2 Channel A DMA Error	Level	
145	SPORT2_B_DMA_ERR	SPORT2 Channel B DMA Error	Level	
146	SPORT3_A_DMA_ERR	SPORT3 Channel A DMA Error	Level	
147	SPORT3_B_DMA_ERR	SPORT3 Channel B DMA Error	Level	
148	SPORT4_A_DMA_ERR	SPORT4 Channel A DMA Error	Level	
149	SPORT4_B_DMA_ERR	SPORT4 Channel B DMA Error	Level	
150	SPORT5_A_DMA_ERR	SPORT5 Channel A DMA Error	Level	
151	SPORT5_B_DMA_ERR	SPORT5 Channel B DMA Error	Level	

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
152	SPORT6_A_DMA_ERR	SPORT6 Channel A DMA Error	Level	
153	SPORT6_B_DMA_ERR	SPORT6 Channel B DMA Error	Level	
154	SPORT7_A_DMA_ERR	SPORT7 Channel A DMA Error	Level	
155	SPORT7_B_DMA_ERR	SPORT7 Channel B DMA Error	Level	
156	SPI2_TXDMA_ERR	SPI2 TX DMA Channel Error	Level	
157	SPI2_RXDMA_ERR	SPI2 RX DMA Channel Error	Level	
158	SPI0_TXDMA_ERR	SPI0 TX DMA Channel Error	Level	
159	SPI0_RXDMA_ERR	SPI0 RX DMA Channel Error	Level	
160	SPI1_TXDMA_ERR	SPI1 TX DMA Channel Error	Level	
161	SPI1_RXDMA_ERR	SPI1 RX DMA Channel Error	Level	
162	UART0_TXDMA_ERR	UART0 Transmit DMA Error	Level	
163	UART0_RXDMA_ERR	UART0 Receive DMA Error	Level	
164	UART1_TXDMA_ERR	UART1 Transmit DMA Error	Level	
165	UART1_RXDMA_ERR	UART1 Receive DMA Error	Level	
166	UART2_TXDMA_ERR	UART2 Transmit DMA Error	Level	
167	UART2_RXDMA_ERR	UART2 Receive DMA Error	Level	
168	LP0_DMA_ERR	LP0 DMA Data Error		
169	LP1_DMA_ERR	LP1 DMA Data Error		
170	MDMA0_SRC_ERR	Standard BW MDMA0 Channel 0 (Source) Error		
171	MDMA0_DST_ERR	Standard BW MDMA0 Channel 1 (Destination) Error		
172	MDMA1_SRC_ERR	Standard BW MDMA1 Channel 0 (Source) Error		
173	MDMA1_DST_ERR	Standard BW MDMA1 Channel 1 (Destination) Error		
174	MDMA2_SRC_ERR	Enhanced BW MDMA2 Channel 0 (Source) Error		
175	MDMA2_DST_ERR	Enhanced BW MDMA2 Channel 1 (Destination) Error		
176	MDMA3_SRC_ERR	Max BW MDMA3 Channel 0 (Source) Error		

Table 6-5: ADSP-2156x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
177	MDMA3_DST_ERR	Max BW MDMA3 Channel 1 (Destination) Error		
179	SWU1_EVT	SWU1 Event L2 Memory DMA Port 0	None	
180	SWU2_EVT	SWU2 Event L2 Memory Core Port 0	None	
181	SWU7_EVT	SWU7 Event, Core ID=1, Slave Port 1	None	
182	SWU8_EVT	SWU8 Event, Core ID=1, Slave Port 2	None	
183	SWU11_EVT	SWU11 Event SMMR	None	
184	SWU12_EVT	SWU12 Event SPI2 and SPI3	None	
185	SWU13_EVT	SWU13 Event DMC0	None	
186	SPU0_INT	SPU0 Interrupt	Level	
187	SMPU_AGGR_INT	SMPU Aggregated Interrupt/Event		
188	SYS_C0_DS_SLV_ACC_INTR	Core Slave access interrupt		
189	DAI0_GBL_SPORT_INT0	DAI0 DAI0 Global SPORT interrupt0	None	
190	DAI0_GBL_SPORT_INT1	DAI0 DAI0 Global SPORT interrupt1	None	
191	DAI1_GBL_SPORT_INT0	DAI1 DAI1 Global SPORT interrupt0	None	
192	DAI1_GBL_SPORT_INT1	DAI1 DAI1 Global SPORT interrupt1	None	

SEC Definitions

The event controller uses the following definitions.

System Events

System source indications including interrupts and faults.

System Source

Point of origin of system event.

SID (Identification, unique)

Source numeric identifier for each system source connected to the SEC.

SSI

SEC source interface, system event source control, and status subblock of the SEC.

SCI

SEC core interface, core interface subblock of the SEC

SPR

SEC prioritizer determines the highest priority pending interrupt and the highest priority active interrupt. The SPR provides these interrupts in the appropriate registers of the SCI for the priority and nesting model of the SCI.

SFI

SEC Fault Interface, fault management subblock of the SEC.

SEC Block Diagram

The *SEC Block Diagram* shows the event management architecture.

System sources connect to the SEC through the SSI. Each core has a dedicated SCI. The SFI provides fault action connections to the rest of the system.

SEC Fault Interface (SFI)

The SFI manages fault events and associated actions. The fault management support provided in the SEC helps satisfy the safety requirements of various applications. The SSI provides the highest priority pending source that is enabled as a fault. The SFI captures this value and enables a countdown, and once the countdown expires, takes the prescribed action.

Fault actions which can be configured, as shown in *SFI Block Diagram*, include

- Trigger Output
- System Reset
- Fault Output
 - Computer Operating Properly (COP) mode
 - Fault Mode

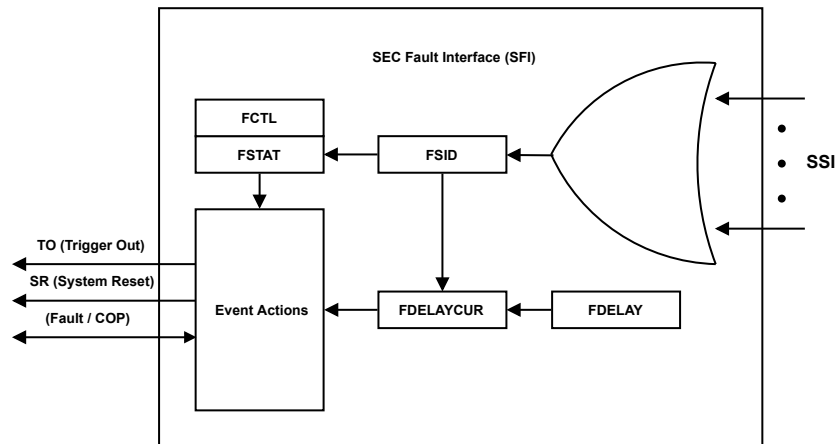


Figure 6-2: SFI Block Diagram

Fault Management

System sources can be enabled as fault sources in the `SEC_SCTL[n]` register. When a source enabled as a fault moves to pending, it is forwarded to the SFI as a fault indication. The pending bit (`SEC_FSTAT.PND`) indicates a source has signaled a fault assertion but it has not yet triggered the event actions (if delay is enabled). The SEC fault interface sets the `SEC_FSTAT.PND` bit when the fault source ID register (`SEC_FSID`) is updated on assertion of a fault source input. The system source pending triggers a fault pending and after a programmable delay the fault moves to active. Event actions then execute if appropriate action is not taken by the core. The `SEC_FSTAT.ACT` bit indicates that the SEC has received a fault source input, the delay has expired, and the fault actions are enabled.

The `SEC_FSTAT.NPND` bit indicates if one or more sources have signaled a fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the `SEC_FSTAT.NPND` bit when the fault interface detects assertion of any enabled fault source input, while either the `SEC_FSTAT.PND` or `SEC_FSTAT.ACT` bits are set. The SEC clears the `SEC_FSTAT.NPND` bit when there are no fault sources waiting.

A fault indication from an external device can also be detected on sampling the fault signals. When a fault is detected the `SEC_FSTAT.ACT` and `SEC_FSID.FEXT` bits are set. The assertion of either signal results in a fault input detection.

The `SEC_FEND` register receives a fault end indication from the core. The core writes the SID of the fault to the `SEC_FEND` register. If the SID matches the value in the `SEC_FSID` register, the `SEC_FSTAT.PND` and `SEC_FSTAT.ACT` bits are cleared.

SEC Core Interface (SCI)

The SCI manages communication between the corresponding core and the SEC. The SEC prioritizer (SPR) of the SCI receives pending, active, and priority information from the SSI for each system event source assigned to this SCI. The SPR determines the highest-priority pending system event and the SCI determines whether it propagates to the core. The SCI maintains the coherency for the system event service model implemented on the connected core.

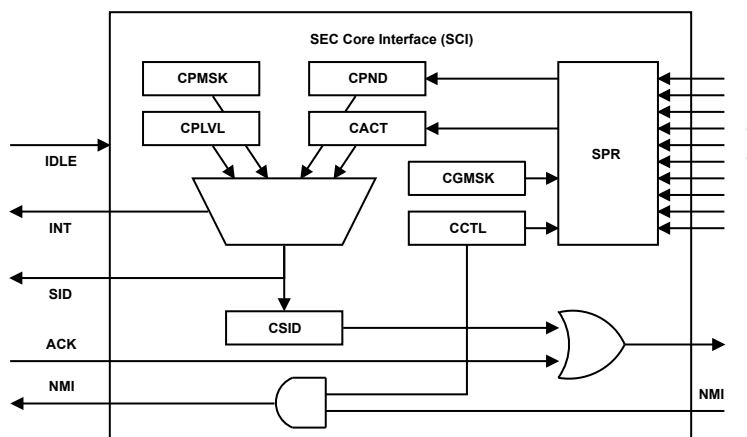


Figure 6-3: SCI Overview Block Diagram

SEC Source Interface (SSI)

The SSI manages all of the system event sources. It maintains the status of each source in the corresponding `SEC_SSTAT[n]` register. The corresponding `SEC_SCTL[n]` register manages the control of each source. A pending and enabled event passes its indication and priority to the SCI to which it is assigned for further processing.

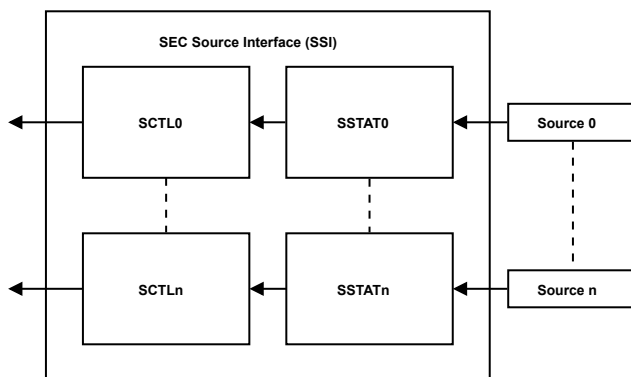


Figure 6-4: SSI Overview Block Diagram

SEC Architectural Concepts

The following sections describe SEC architectural features.

System Interrupt Acknowledge

A system interrupt acknowledge occurs when the core provides an indication that it has acquired the SID of the interrupt last issued by the SEC. The SEC core interface option allows generation by:

- A slave port write to the `SEC_CSID[n]` register.
- The assertion of an input acknowledge signal (the connected core generates the signal).

System Interrupt Groups

System sources can be assigned to groups using the `SEC_SCTL[n].GRP` bit field. Source groups allow fast context switching for system interrupts at each SCI. The `SEC_CGMSK[n]` register allows quick masking of interrupt groups of unlimited size with a single write operation.

System Interrupt Flow

An enabled and asserted system interrupt source is latched at the SSI and routed to the appropriate SCI based on the core target select (`SEC_SCTL[n].CTG`) bit field setting. The SEC priority ordering determines the highest priority pending system interrupt and the SCI updates the `SEC_CPND[n].SID` and `SEC_CACT[n].PRIO` bit field values. The SCI compares the `SEC_CPND[n]` register value against the highest priority active source in the `SEC_CACT[n]` register).

The priority level register (`SEC_CPLVL[n]`) determines how many of the MSBs the SEC uses in the comparison. The priority mask register (`SEC_CPMSK[n]`) and the group mask register (`SEC_CGMSK[n]`) determines which pending interrupt sources participate. If the `SEC_CPND[n]` register value is a higher priority (lower value) than the priority of the `SEC_CACT[n]` register from the comparison based on the `SEC_CPLVL[n]` register, the system interrupt output is asserted. The source ID register (`SEC_CSID[n]`) is updated with the `SEC_CPND[n].SID` bit field value and forwarded to the connected core.

After the core provides an interrupt acknowledgment, the interrupt source is active, until the SEC completes interrupt service with a write to the `SEC_END.SID` bit field with the same value. Note the following:

- Interrupt acknowledgement occurs with an MMR write of the `SEC_CSID[n]` register or the core version of the `SEC_CSID[n]` register.
- Interrupt active status indication is `SEC_SSTAT[n].ACT==1`.

The following sequence shows the example flow for a single interrupt.

1. The SEC compares the `SEC_CPND[n]` register value to the `SEC_CACT[n]` register value. If the interrupt in the `SEC_CPND[n]` register is higher priority, continue.
2. The SEC copies the `SEC_CPND[n]` register value to the `SEC_CSID[n]` register and asserts the interrupt signal.
3. The core reads the `SEC_CSID[n]` register (or core version).
4. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the interrupt signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source going active.
6. The core writes the `SEC_CSID[n]` of the active interrupt to the `SEC_END` register.
7. The SEC clears the `SEC_SSTAT[n].ACT` bit of the source being ended.

The following sequence shows the example flow for interrupt nesting where interrupt A is a lower priority and occurs earlier than interrupt B.

1. The SEC compares the `SEC_CPND[n]` (A) register value to the `SEC_CACT[n]` register and if the interrupt in the `SEC_CPND[n]` register is a higher priority, continue.
2. The SEC copies `SEC_CPND[n]` (A) register to the `SEC_CSID[n]` register and asserts the interrupt signal.
3. The core reads the `SEC_CSID[n]` (A) register (or core version).
4. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the INT signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source (A) going active.
6. The SEC compares the `SEC_CPND[n]` (B) register value to the `SEC_CACT[n]` (A) register value. If the `SEC_CACT[n]` (B) register value is a higher priority, continue.
7. The SEC copies the `SEC_CPND[n]` (B) register value to `SEC_CSID[n]` register and asserts the interrupt signal.
8. The core reads the `SEC_CSID[n]` (B) register (or core version).
9. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
10. The SEC deasserts the INT signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source (B) going active.
11. The core writes the `SEC_CSID[n]` of the active interrupt (B) to the `SEC_END` register.
12. The SEC clears the `SEC_SSTAT[n].ACT` bit of the source (B) being ended.
13. The core writes the `SEC_CSID[n]` of the active interrupt (A) to the `SEC_END` register.
14. The SEC clears the `SEC_SSTAT[n].ACT` bit of the source (A) being ended.

System Interrupt Priorities

Each system interrupt source has its own programmable priority level which is configured using the `SEC_SCTL[n].PRIO` bit field. The SCI evaluates the priority of all pending sources to determine the source of the highest-priority pending system interrupt for forwarding to the attached core. If more than one source of the pending system interrupt has the same priority setting, the SCI chooses the one with the lowest SID. For example, if SID 0, SID 1, and SID 2 are all pending and have the same priority setting, the SCI chooses SID 0 as the highest-priority source.

SEC Error

The processor includes an SEC error (`SEC_GSTAT.ERR`) as a source input to the SEC to allow for handling the error as an interrupt or fault.

SEC Programming Model

Implementing a system interrupt service model using the SEC requires, at a minimum:

- Proper configuration of a system interrupt source (for example a peripheral or DMA)
- A core interrupt or event service model

The core must be configured for response to system interrupts from the SEC. The SEC must be configured to enable and map the system interrupt source to the correct SCI and to forward interrupts to the connected core.

The system interrupt source must be configured to generate interrupt assertions. Alternatively, the processor can use software triggering for interrupt assertion. Software driven interrupts are generated by writing the source ID of the interrupt to be triggered to the `SEC_RAISE` register.

Programming Concepts

The following list provides the basic programming concepts necessary for configuring the SEC.

- Configuring an SSI as a system interrupt for a specific core.
- Configuring an SCI to provide system interrupts to the connected core (See [Configuring a System Source to Interrupt a Core](#)).
- Configuring an SSI as a system fault (See [Configuring a System Source as a Fault](#)).
- Configuring the SFI to manage system faults.

Programming Examples

This section provides example programming tasks that are typical for SEC usage.

Fault Management Interface Programming Model

The SFI interface can be programmed to manage fault events from system sources and associated actions such as issuing a system reset when watchdog expiration event occurs.

1. Set the `SEC_GCTL.EN` bit to enable the SEC.
2. Write to the `SEC_FCTL` register to configure specific fault actions.
 - Trigger Output. Set the `SEC_FCTL.TOEN` bit for the SEC to produce trigger outputs when a fault becomes active. The `SEC_FCTL.TES` bit can be programmed to select the event that directs the SEC to assert trigger output when a fault is pending or active. Configure slaves for SEC fault trigger master output.

NOTE: If the `SEC_FCTL.TOEN` and or the `SEC_FCTL.TES` bits =1 (Trigger Output Enabled and Trigger on Fault Pending), an external fault (if enabled by the `SEC_FCTL.FIEN` bit) will not issue a trigger since Fault Pending is bypassed for external faults.

 - System Reset. The Reset Control Unit (RCU) controls how the functional units enter and exit reset. Configure the `RCU_CTL.SRSTREQEN` bit. This bit controls whether the sources of reset are enabled to perform a system reset. To issue a system reset request when a fault becomes active, set the `SEC_FCTL.SREN` bit. The SEC fault system reset delay register (`SEC_FSRDLY`) can be programmed for the delay, if required, from a fault becoming active to system reset request assertion.

- **Fault Output.** This configuration allows the SEC to indicate the fault status based on the `SEC_FCTL.CMS` bit configuration.
 - **Computer Operating Normally (COP) mode.** To configure fault output for COP mode, set the `SEC_FCTL.FOEN` bit to enable fault output. Set the `SEC_FCTL.CMS` bit to select COP mode to toggle the fault pin when no fault is active. Program the `SEC_FCOPP` period register with a desired width value for the COP toggled output pin.
 - **Fault mode.** Set the `SEC_FCTL.FOEN` bit to enable fault output. The `SEC_FCTL.CMS` bit should be set to Fault mode to toggle the fault pin when a fault is active.
3. If required, program the Fault Input to sample fault inputs from external devices on fault pins. Configure the `SEC_FCTL.FIEN` bit to enable the SEC to sample a fault input from an external device.

ADDITIONAL INFORMATION: The `SEC_FCTL.FIEN` bit should be set only while the `SEC_FCTL.EN` bit is low. If the `SEC_FCTL.EN` bit is already high and the `SEC_FCTL.FIEN` bit needs to be set, the `SEC_FCTL.EN` bit should be cleared first. Fault input can only be enabled when Fault mode is selected by the `SEC_FCTL.CMS` bit.

4. Program the required fault delay to the `SEC_FDLY.COUNT` bit field if a delay between fault source assertion and the fault response is required.
5. Configure the `SEC_FCTL` register to enable the SEC.

ADDITIONAL INFORMATION: The `SEC_FCTL.EN` bit should be set only while the `SEC_FSTAT.ACT` bit is low.

6. Write to the control register of a specific source register using the `SEC_SCTL[n]` register to enable the source as a fault.

Configuring a System Source to Interrupt a Core

To configure a system source to interrupt a core, the SEC itself must be enabled with the source interface (SSI) and core interface (SCI) properly initialized. Specifically, the SCI must be set up to accept interrupt signaling from the SEC and pass them to the specified core, and the SSI must properly enable each of the peripheral interrupt sources to generate interrupt signals and optionally define a priority scheme that overrides the default priority settings. In summary:

1. Write to the `SEC_GCTL` register to enable the SEC.
2. Write to the appropriate SCI `SEC_CCTL[n]` register to enable SEC interrupts to be sent to that core.
3. Write to the appropriate SSI `SEC_SCTL[n]` register to enable that peripheral as an interrupt source and to set the core target field to map the source to the desired SCI.
4. (Optional) By default, all the SEC interrupts are grouped as a single priority level, so passing of peripheral interrupt requests from the SEC is based solely on the default enumerated source ID. By programming the `SEC_CPLVL[n].PLVL` register, interrupt sources can be grouped into priority levels within the SEC such that arbitration is first performed by source ID within a grouped priority level before proceeding to the next

priority level, thus providing the flexibility to have lower-priority interrupt sources considered before higher-priority sources.

ADDITIONAL INFORMATION: The `SEC_CPMSK[n]` and `SEC_CGMSK[n]` registers must also can be programmed to mask the interrupts based on the customized levels and grouping.

Core/SEC Handshake Requirements to Ensure Proper Interrupt Handling

A specific handshake with the SEC is required to handle interrupts associated with an individual core. The handshake ensures that nested interrupts are properly tracked and that new peripheral interrupts being raised within the SEC are either passed immediately to the core or held off and queued within the SEC for later servicing.

Use the following procedure to write a custom dispatcher inside the Interrupt Service Routine. Note that the core needs to read and acknowledge the `SEC_CSID[n]` register by writing the same value. The core must also write to the `SEC_END` register after the ISR execution completes.

1. Read the `SEC_CSID[n]` register to obtain the source ID of the peripheral interrupt request.
2. Write the read value back to the `SEC_CSID[n]` register to send the acknowledge signal to the SEC that the core has accepted and begun processing the interrupt request.
3. Execute the actual ISR (typically a call to a specific handler function from a look-up table based on the peripheral source ID). Write to the `SEC_GCTL` register to enable the SEC.
4. Write the value of the `SEC_CSID[n]` register of the active interrupt (read in step 1 above) to the `SEC_END` register to signal to the SEC that the interrupt has now been serviced.
5. Return from interrupt.

This procedure allows a higher-priority interrupt raised by the SEC to be serviced by the core after step 2. The SEC knows what it passed to the core because of the write to the `SEC_CSID[n]` register. After the core acknowledges that write, the SEC knows whether or not newly raised peripheral interrupts are a higher priority than the highest-priority interrupt currently being processed by the core.

- If higher priority, the SEC pushes the current `SEC_CSID[n]` value to an internal stack, writes the new `SEC_CSID[n]` value, and asserts a new SEC interrupt request.
- If lower priority, the SEC queues the interrupt until the core writes to the `SEC_END` register with the source ID of the higher-priority interrupt, confirming that it was fully processed.

At this point the `SEC_CSID[n]` value is popped from the internal stack and any pending peripheral interrupt requests are arbitrated before the SEC writes the new `SEC_CSID[n]` value and asserts a new interrupt request. At the same time the core self-nests the latched SEC interrupt requests as needed. When a higher-priority interrupt is presented to the core the write to the `SEC_END` register in the SEC handler epilog code guarantees that each nested level has the required handshake to signal to the SEC block that each individual source ID interrupt request is fully serviced. See the *SHARC+ Core Programming Reference* for more details about SEC handler code.

Configuring a System Source as a Fault

Use the following procedure configure a system source as a fault.

1. Write to the `SEC_GCTL` register to enable the SEC.
2. Write to the `SEC_FCTL` register to configure specific fault actions.
3. Write to the `SEC_FDLY` bit field to specify fault delay.
4. Write to the control register of a specific source to enable the source as a fault.

SEC Programming Restrictions

Setting the `SEC_FCTL.EN` bit while the `SEC_FSTAT.ACT` bit is high can result in unpredictable behavior. To avoid this issue, set the `SEC_FCTL.EN` bit while the `SEC_FSTAT.ACT` bit is low. The `SEC_FSTAT.ACT` bit is only set when the `SEC_FCTL.EN` bit is high. Therefore, the problem can only occur if the `SEC_FCTL.EN` bit transitions from 1 to 0 and then to 1 again.

Writing to `SEC_FEND` to end a fault with both the `SEC_FCTL.FOEN` bit and the `SEC_FCTL.FIEN` bit set can result in erroneous external fault detection. If this operation (ending a fault) and configuration (fault input and fault output enabled) are required by the application, clear the `SEC_FCTL.FOEN` bit prior to writing to `SEC_FEND`. The recommended sequence for ending a fault with the `SEC_FCTL.FIEN` or `SEC_FCTL.FOEN==1` is as follows:

1. Clear the `SEC_FCTL.FOEN` bit.
2. Write to the `SEC_FEND` register.
3. Set the `SEC_FCTL.FOEN` bit.

ADSP-2156x SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 6-6: ADSP-2156x SEC Register List

Name	Description
<code>SEC_CACT[n]</code>	SCI Active Register n
<code>SEC_CCTL[n]</code>	SCI Control Register n
<code>SEC_CGMSK[n]</code>	SCI Group Mask Register n
<code>SEC_CPLVL[n]</code>	SCI Priority Level Register n
<code>SEC_CPMSK[n]</code>	SCI Priority Mask Register n
<code>SEC_CPND[n]</code>	Core Pending Register n
<code>SEC_CSID[n]</code>	SCI Source ID Register n
<code>SEC_CSTAT[n]</code>	SCI Status Register n

Table 6-6: ADSP-2156x SEC Register List (Continued)

Name	Description
SEC_END	Global End Register
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

SCI Active Register n

The SEC SCI active interrupt register (`SEC_CACT[n]`) contains the source ID and priority of the highest priority active interrupt detected by the SEC prioritizer.

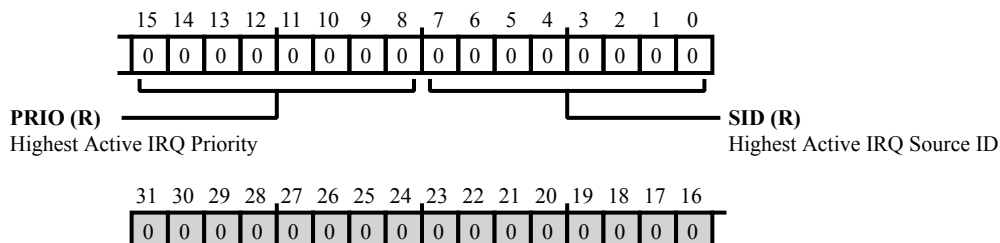


Figure 6-5: SEC_CACT[n] Register Diagram

Table 6-7: SEC_CACT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Active IRQ Priority. The <code>SEC_CACT[n].PRIO</code> indicates the priority value of the highest priority active interrupt for core n.
7:0 (R/NW)	SID	Highest Active IRQ Source ID. The <code>SEC_CACT[n].SID</code> identifies the source ID value of the highest priority active interrupt for core n.

SCI Control Register n

The SEC control register (`SEC_CCTL[n]`) contains SCI control bits for all system sources.

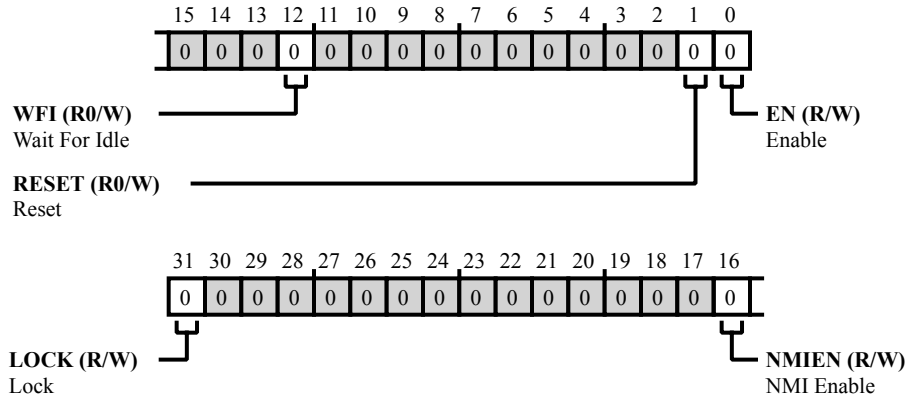


Figure 6-6: SEC_CCTL[n] Register Diagram

Table 6-8: SEC_CCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If the global lock is enabled (<code>SPU_CTL . GLCK</code> bit =1) and the <code>SEC_CCTL[n] . LOCK</code> bit is enabled, the <code>SEC_CCTL[n]</code> register is read only.
		0 Unlock 1 Lock
16 (R/W)	NMIEN	NMI Enable.
		The <code>SEC_CCTL[n] . NMIEN</code> bit controls NMI propagation to the core. When the <code>SEC_CCTL[n] . NMIEN</code> bit is enabled, the SCI allows NMIs to propagate to the core for servicing.
		0 Disable 1 Enable
12 (R0/W)	WFI	Wait For Idle.
		When set, the <code>SEC_CCTL[n] . WFI</code> bit forces the SCI to wait for indication of core idle before the SCI resumes activity.
		0 No Action 1 Wait for Idle

Table 6-8: SEC_CCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R0/W)	RESET	Reset. When set, the SEC_CCTL[n].RESET bit resets all SCI registers to their default values.
		0 No Action
		1 Reset
0 (R/W)	EN	Enable. The SEC_CCTL[n].EN bit controls operation of the SCI. Clearing the SEC_CCTL[n].EN bit halts the execution of the SCI without resetting status registers. (The INT signal to a core is not affected.) Setting the SEC_CCTL[n].EN bit enables the SCI to begin or resume operation with the current configuration and status.
		0 Disable
		1 Enable

SCI Group Mask Register n

The SEC SCI group mask register (`SEC_CGMSK[n]`) contains selections for a group mask, an ungroup mask, and a register lock. This register contains the system interrupt group masks for the connected core. The core uses the `SEC_CGMSK[n].UGRP` and `SEC_CGMSK[n].GRP` fields to mask (disable) interrupts from the specified groups.

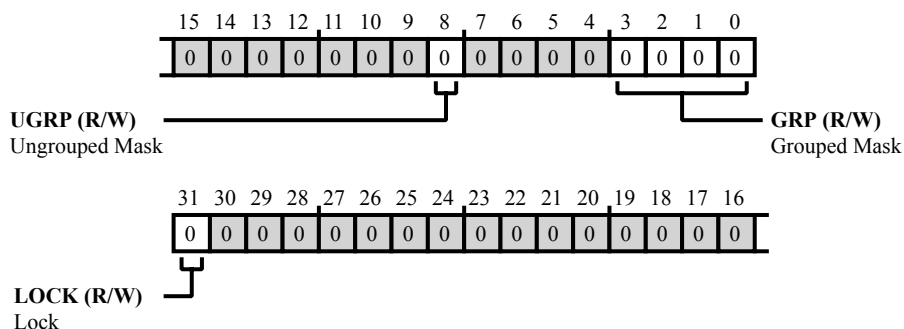


Figure 6-7: SEC_CGMSK[n] Register Diagram

Table 6-9: SEC_CGMSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>SEC_CGMSK[n].LOCK</code> bit is enabled, the <code>SEC_CGMSK[n]</code> register is read only.
		0 Unlock
		1 Lock
8 (R/W)	UGRP	Ungrouped Mask. The <code>SEC_CGMSK[n].UGRP</code> bit masks interrupts (if set) for the ungrouped interrupt sources for core n.
		0 Unmask Ungrouped Sources
		1 Mask Ungrouped Sources
3:0 (R/W)	GRP	Grouped Mask. The <code>SEC_CGMSK[n].GRP</code> field selects a group of interrupt sources to mask for core n. (For more information about interrupt source groups, see the <code>SEC_SCTL[n]</code> register description.)
		0 No groups masked
		1 Mask group 0
		2 Mask group 1
		3 Mask groups 0, 1

Table 6-9: SEC_CGMSK[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	Mask group 2
		5	Mask groups 0, 2
		6	Mask groups 1, 2
		7	Mask groups 0, 1, 2
		8	Mask group 3
		9	Mask groups 0, 3
		10	Mask groups 1, 3
		11	Mask groups 0, 1, 3
		12	Mask groups 2, 3
		13	Mask groups 0, 2, 3
		14	Mask groups 1, 2, 3
		15	Mask groups 0, 1, 2, 3

SCI Priority Level Register n

The SEC SCI priority level register (`SEC_CPLVL[n]`) contains selections for priority levels and a register lock. This register is used to divide the total number of priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same level.

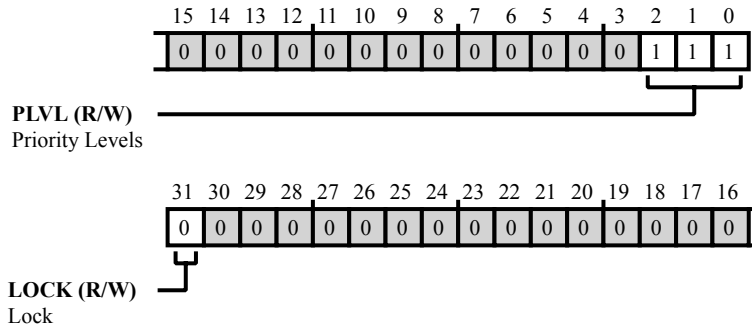


Figure 6-8: SEC_CPLVL[n] Register Diagram

Table 6-10: SEC_CPLVL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If the global lock is enabled (<code>SPU_CTL . GLCK</code> bit =1) and the <code>SEC_CPLVL[n] . LOCK</code> bit is enabled, the <code>SEC_CPLVL[n]</code> register is read only.
		0 Unlock 1 Lock
2:0 (R/W)	PLVL	Priority Levels.
		The <code>SEC_CPLVL[n] . PLVL</code> field serves to divide the total number of interrupt priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same interrupt level. The sub-level priority value specifies the number of MSBs (minus 1) designated to interrupt levels, while the remaining LSBs are designated for sub-level specification.
		For example, if the <code>SEC_CPLVL[n] . PLVL</code> field is set to two, the result is four priority levels are specified, because only the two MSBs are used for preemption evaluation. The remaining bits of the priority setting are used for sub-level prioritization.
		0 1 MSBs (2 priority levels)
		1 2 MSBs (4 priority levels)
		2 3 MSBs (8 priority levels)
		3 4 MSBs (16 priority levels)
4 5 MSBs (32 priority levels)		
5 6 MSBs (64 priority levels)		

Table 6-10: SEC_CPLVL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	7 MSBs (128 priority levels)
		7	8 MSBs (256 priority levels)

SCI Priority Mask Register n

The SEC SCI priority mask register (`SEC_CPMSK[n]`) contains the SCI priority mask for core n and includes a register lock.

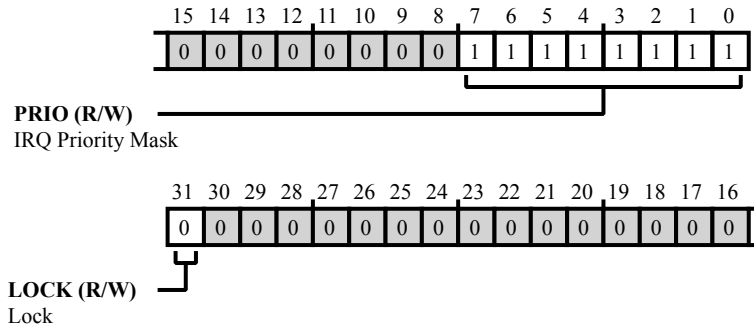


Figure 6-9: SEC_CPMSK[n] Register Diagram

Table 6-11: SEC_CPMSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>SEC_CPMSK[n].LOCK</code> bit is enabled, the <code>SEC_CPMSK[n]</code> register is read only.
		0 Unlock
		1 Lock
7:0 (R/W)	PRIO	IRQ Priority Mask. The <code>SEC_CPMSK[n].PRIO</code> contains the system interrupt priority mask for core n. The core uses the <code>SEC_CPMSK[n].PRIO</code> field to mask (block) interrupts below the specified level.
		0 Priority level 0 (highest)
		1-254
		255 Priority level 255 (lowest)

Core Pending Register n

The SCI pending interrupt register (`SEC_CPND[n]`) contains the source ID and priority of the highest priority pending interrupt detected by the SEC prioritizer.

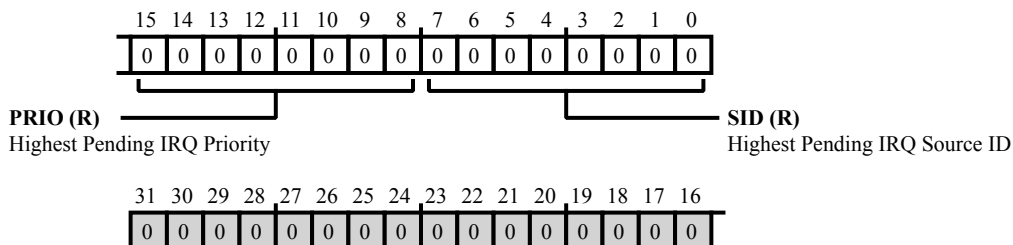


Figure 6-10: SEC_CPND[n] Register Diagram

Table 6-12: SEC_CPND[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Pending IRQ Priority. The <code>SEC_CPND[n].PRIO</code> indicates the priority value of the highest priority pending interrupt for core n.
7:0 (R/NW)	SID	Highest Pending IRQ Source ID. The <code>SEC_CPND[n].SID</code> identifies the source ID value of the highest priority pending interrupt for core n.

SCI Source ID Register n

The SCI source ID register (`SEC_CSID[n]`) contains the source ID of the interrupt last issued to core n. The `SEC_CSID[n]` register value is loaded by the SCI when a system interrupt indication is sent to core n. The SCI does not change the `SEC_CSID[n]` until after the interface receives an interrupt acknowledge from core n. Writing to the `SEC_CSID[n]` register generates an interrupt acknowledge, but does not update the value in the register.

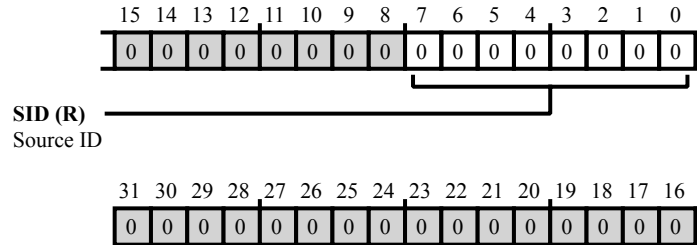


Figure 6-11: SEC_CSID[n] Register Diagram

Table 6-13: SEC_CSID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	SID	Source ID. The <code>SEC_CSID[n].SID</code> bit is the source ID of the interrupt last issued to core n.

SCI Status Register n

The SCI status register (`SEC_CSTAT[n]`) contains status bits, indicating the operational status of the SCI.

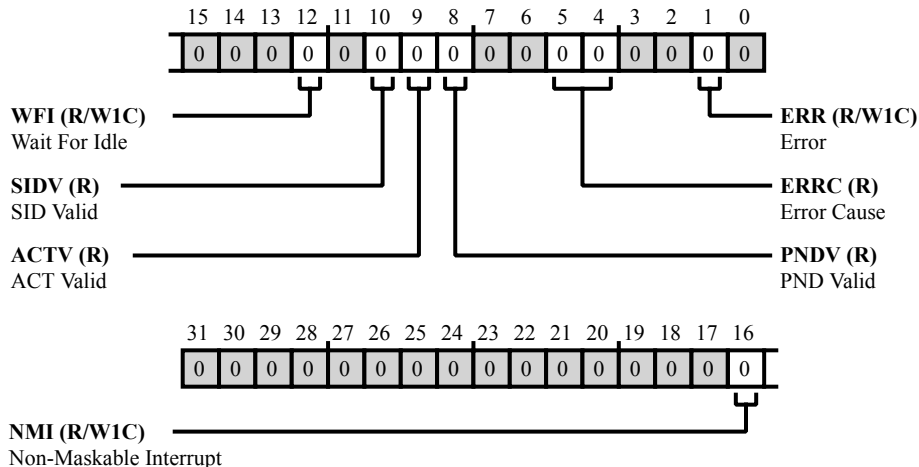


Figure 6-12: SEC_CSTAT[n] Register Diagram

Table 6-14: SEC_CSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W1C)	NMI	Non-Maskable Interrupt. The <code>SEC_CSTAT[n].NMI</code> bit indicates whether an NMI has occurred since the bit was last cleared.
		0 No NMI Occurred
		1 NMI Occurred
12 (R/W1C)	WFI	Wait For Idle. The <code>SEC_CSTAT[n].WFI</code> bit indicates (if set) that the SCI is temporarily disabled, pending a core idle indication. This bit is set when <code>SEC_CCTL[n].WFI</code> is set.
		0 Not Waiting
		1 Waiting
10 (R/NW)	SIDV	SID Valid. The <code>SEC_CSTAT[n].SIDV</code> bit indicates (if set) that the current value in the <code>SEC_CSID[n]</code> register is valid. The SCI sets the <code>SEC_CSTAT[n].SIDV</code> bit when the updating the <code>SEC_CSID[n]</code> register with a new value. The <code>SEC_CSTAT[n].SIDV</code> bit is cleared when the <code>SEC_CSID[n]</code> register is written. This status indication may be used to extract all pending interrupts in a single interrupt service routine.
		0 Invalid
		1 Valid

Table 6-14: SEC_CSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	ACTV	ACT Valid. The SEC_CSTAT[n].ACTV bit indicates (if set) that the current value in the SEC_CACT[n] register is valid. The SCI sets the SEC_CSTAT[n].ACTV bit when updating the SEC_CACT[n] registers with a new value. The SEC_CSTAT[n].ACTV bit is cleared when the SEC_CSID[n] register is written.
		0 Invalid
		1 Valid
8 (R/NW)	PNDV	PND Valid. The SEC_CSTAT[n].PNDV bit indicates (if set) that the current value in the SEC_CPND[n] register is valid. The SCI sets the SEC_CSTAT[n].PNDV bit when updating the SEC_CPND[n] register with a new value. The SEC_CSTAT[n].PNDV bit is cleared when the SEC_CSID[n] register is written.
		0 Invalid
		1 Valid
5:4 (R/NW)	ERRC	Error Cause. The SEC_CSTAT[n].ERRC bits are updated on assertion of the SEC_CSTAT[n].ERR bit to indicate the SCI error type. SEC_CSTAT[n].ERRC is only updated on the assertion of SEC_CSTAT[n].ERR. Subsequent errors while SEC_CSTAT[n].ERR is asserted do not update SEC_CSTAT[n].ERRC.
		0 Reserved
		1 Acknowledge Error. SCI has received an acknowledge without a pending, unacknowledged interrupt present.
		2 Reserved
		3 Reserved
1 (R/W1C)	ERR	Error. The SEC_CSTAT[n].ERR bit indicates that an error has occurred in the SCI. When SEC_CSTAT[n].ERR is set, the SCI updates the SEC_CSTAT[n].ERRC field to the value of the corresponding error cause.
		0 No Error
		1 Error Occurred

Global End Register

The SEC global end register (`SEC_END`) contains a source ID interrupt service end field (`SEC_END.SID`). When a core has finished servicing an interrupt, the core writes the `SEC_END.SID` field in the `SEC_END` register. This write causes the SEC to clear the `SEC_SSTAT[n].ACT` bit in the `SEC_SSTAT[n]` register of the corresponding interrupt.

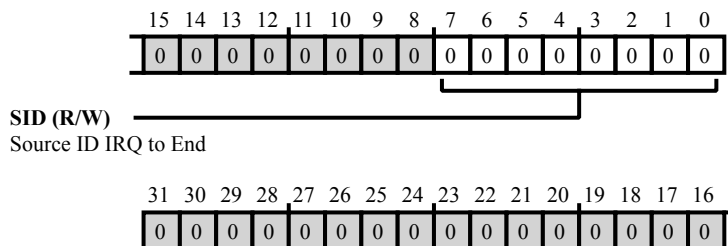


Figure 6-13: SEC_END Register Diagram

Table 6-15: SEC_END Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID IRQ to End. The <code>SEC_END.SID</code> bit field contains the source ID interrupt service end value.

Fault COP Period Register

The SEC fault COP period register (`SEC_FCOPP`) contains the width value (count in (SEC) clock cycles) for the high and low phase of the computer operating properly (COP) toggled output on the COP pin. Note that the actual high/low phase value is the `SEC_FCOPP.COUNT` programmed value plus 1.

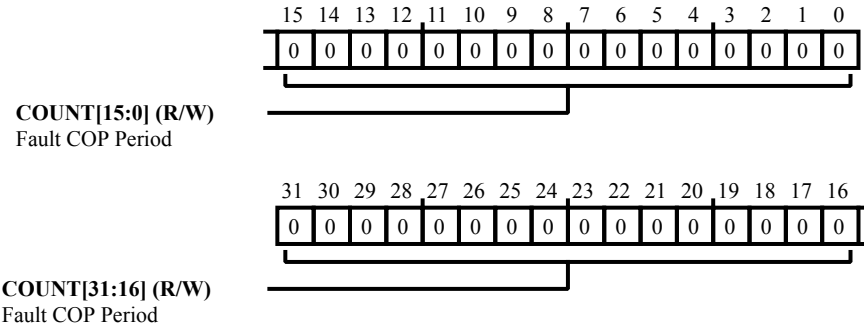


Figure 6-14: SEC_FCOPP Register Diagram

Table 6-16: SEC_FCOPP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault COP Period. The <code>SEC_FCOPP.COUNT</code> bit field is the width value for the high and low phase of the computer operating properly (COP) toggled output on the COP pin.

Fault COP Period Current Register

The SEC fault COP period current register (`SEC_FCOPP_CUR`) contains the active count (in (SEC) clock periods) for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin. The SEC loads the `SEC_FCOPP_CUR` register from the `SEC_FCOPP` register when the `SEC_FCOPP_CUR.COUNT` field is cleared and the SEC is in COP mode (`SEC_FCTL.CMS` bit =1). The SEC decrements the `SEC_FCOPP_CUR` count each (SEC) clock cycle while `SEC_FCTL.CMS` is set and the `SEC_FSTAT.ACT` bit is not set.

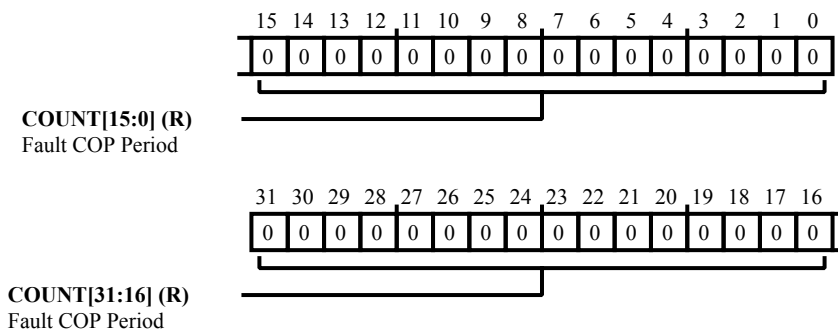


Figure 6-15: SEC_FCOPP_CUR Register Diagram

Table 6-17: SEC_FCOPP_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault COP Period. The <code>SEC_FCOPP_CUR.COUNT</code> bit field is the active count for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin.

Fault Control Register

The SEC fault control register (`SEC_FCTL`) contains fault control bits for all SEC channels. This register controls the operation of the System Fault Management Interface (SFI).

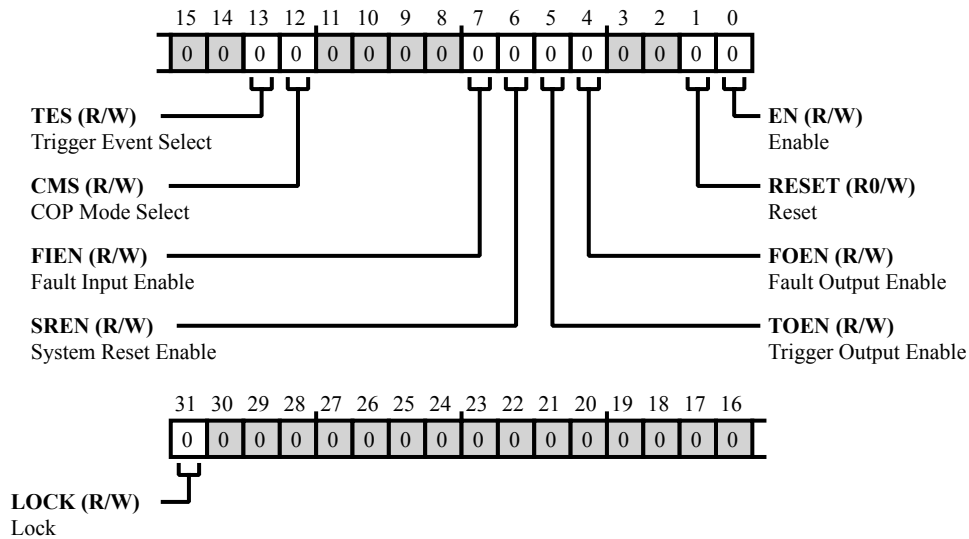


Figure 6-16: SEC_FCTL Register Diagram

Table 6-18: SEC_FCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>SEC_FCTL.LOCK</code> bit is enabled, the <code>SEC_FCTL</code> register is read only.
		0 UnLock
		1 Lock
13 (R/W)	TES	Trigger Event Select. The <code>SEC_FCTL.TES</code> bit selects the event that directs the SEC to assert trigger output. In fault pending mode, the SEC asserts trigger output when a fault is pending. In fault active mode, the SEC asserts trigger output when a fault is active.
		0 Fault Active Mode
		1 Fault Pending Mode

Table 6-18: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	CMS	COP Mode Select. The SEC_FCTL.CMS selects the SEC mode for handling fault input. In COP mode, the SEC toggles the COP pin to indicate that no fault is active and ceases toggling the pin to indicate that a fault is active. In fault mode, the SEC deasserts the fault pin (=0) and fault_b pin (=1) when no fault is active and asserts the fault pin (=1) and fault_b pin (=0) when a fault is active. Not all processors feature both the fault and fault_b pins. Refer to the product data sheet for details.
		0 Fault Mode
		1 COP Mode
7 (R/W)	FIEN	Fault Input Enable. The SEC_FCTL.FIEN bit enables the SEC to sample fault input. If SEC_FCTL.FIEN is set (=1), a fault indication from an external device sets the SEC_FSTAT.ACT bit and SEC_FSID.FEXT bit.
		0 Disable
		1 Enable
6 (R/W)	SREN	System Reset Enable. The SEC_FCTL.SREN bit enables the SEC to issue a system reset request when a fault becomes active.
		0 Disable
		1 Enable
5 (R/W)	TOEN	Trigger Output Enable. The SEC_FCTL.TOEN bit enables the SEC to produce trigger output when a fault becomes active.
		0 Disable
		1 Enable
4 (R/W)	FOEN	Fault Output Enable. The SEC_FCTL.FOEN bit enables the SEC to indicate fault status, according to the SEC_FCTL.CMS bit configuration.
		0 Disable
		1 Enable
1 (R0/W)	RESET	Reset. Setting the SEC_FCTL.RESET bit resets ALL SEC registers to their default values.
		0 No Action
		1 Reset

Table 6-18: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The SEC_FCTL.EN bit controls the operational state of the SEC. Clearing the SEC_FCTL.EN bit halts the execution of the SEC without resetting status registers. Setting the SEC_FCTL.EN bit enables the SEC to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Fault Delay Register

The SEC fault delay register (`SEC_FDLY`) contains the number (`SEC_FDLY.COUNT` field) of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

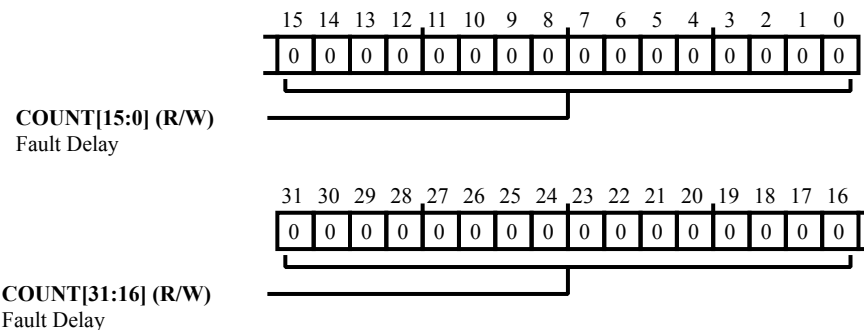


Figure 6-17: SEC_FDLY Register Diagram

Table 6-19: SEC_FDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault Delay. The <code>SEC_FDLY.COUNT</code> bit field is the number of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

Fault Delay Current Register

The SEC fault delay current register (`SEC_FDLY_CUR`) contains the active count (`SEC_FDLY_CUR.COUNT` field) in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled. The count is loaded from the `SEC_FDLY` register when a fault becomes pending (`SEC_FSTAT.PND` bit is set). The SEC decrements the value in `SEC_FDLY_CUR` each (SEC) clock cycle while the `SEC_FSTAT.PND` bit is set.

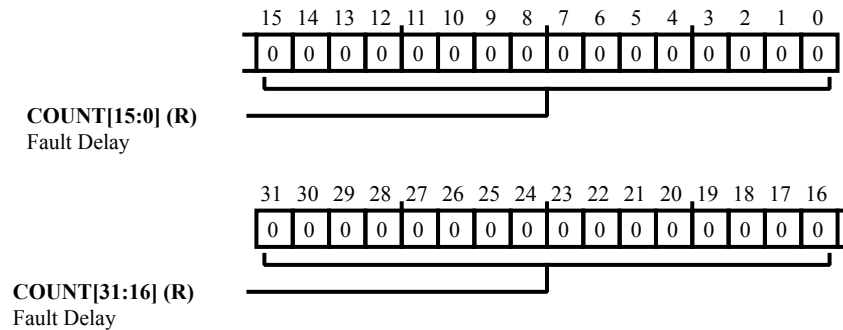


Figure 6-18: SEC_FDLY_CUR Register Diagram

Table 6-20: SEC_FDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault Delay. The <code>SEC_FDLY_CUR.COUNT</code> bit field is the active count in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled.

Fault End Register

The SEC fault end register (`SEC_FEND`) contains fault source ID and internal/external fields. This register receives fault end indication from a core.

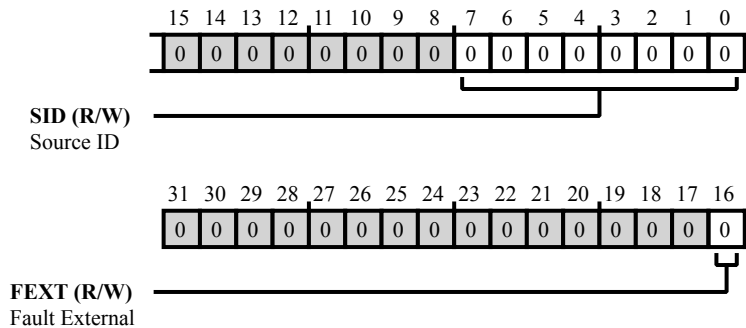


Figure 6-19: SEC_FEND Register Diagram

Table 6-21: SEC_FEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	FEXT	Fault External. Setting the <code>SEC_FEND.FEXT</code> bit, when the <code>SEC_FEND.SID</code> field is cleared, clears an active fault from an external source.
		0 Fault Internal
		1 Fault External
7:0 (R/W)	SID	Source ID. The <code>SEC_FEND.SID</code> identifies a fault to be ended as indicated to the SEC by the core. The core loads the <code>SEC_FEND.SID</code> field value. If the <code>SEC_FEND.SID</code> value matches the <code>SEC_FSID.SID</code> value, the <code>SEC_FSTAT.PND</code> bit and <code>SEC_FSTAT.ACT</code> bit are cleared.

Fault Source ID Register

The SEC fault source ID register (`SEC_FSID`) contains a fault source ID and internal/external fields.

NOTE: These bits are not reset by system reset so that a fault that automatically triggers a system reset to avoid a fault may be analyzed after the reset occurs.

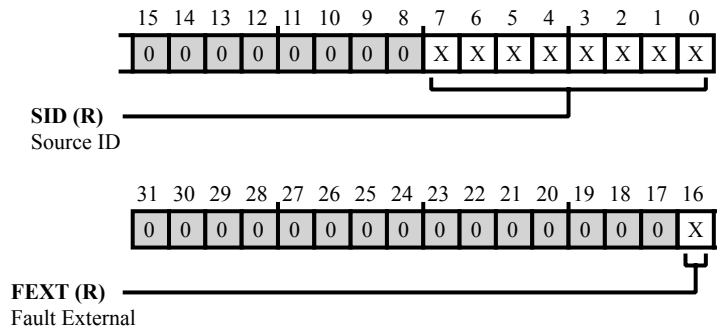


Figure 6-20: SEC_FSID Register Diagram

Table 6-22: SEC_FSID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/NW)	FEXT	Fault External. The <code>SEC_FSID.FEXT</code> bit indicates that the last active fault was asserted by an external device. The SEC sets the <code>SEC_FSID.FEXT</code> bit when the <code>SEC_FSTAT.ACT</code> bit is set by the fault input pins. The <code>SEC_FSID.FEXT</code> bit is cleared when the <code>SEC_FSTAT.ACT</code> bit is set by an internal fault or when the external fault is ended. When the <code>SEC_FSID.FEXT</code> bit is set, the <code>SEC_FSID.SID</code> is cleared.
		0 Fault Internal
		1 Fault External
7:0 (R/NW)	SID	Source ID. The <code>SEC_FSID.SID</code> identifies the fault assertion detected by the SEC fault interface. The SEC loads the <code>SEC_FSID.SID</code> field value when a system fault indication is asserted. The SEC fault interface does not change the <code>SEC_FSID.SID</code> value until the fault is no longer pending or active, as indicated by the <code>SEC_FSTAT.PND</code> bit and <code>SEC_FSTAT.ACT</code> bit being cleared in the <code>SEC_FSTAT</code> register.

Fault System Reset Delay Register

The SEC fault system reset delay register (`SEC_FSRDLY`) contains the number (`SEC_FSRDLY.COUNT` field) of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion, if enabled.

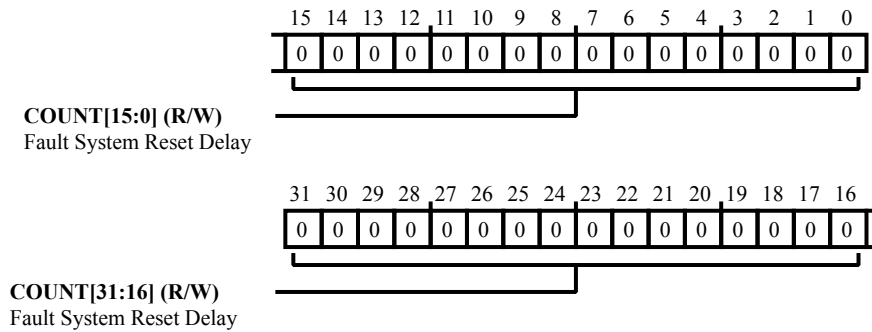


Figure 6-21: SEC_FSRDLY Register Diagram

Table 6-23: SEC_FSRDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault System Reset Delay. The <code>SEC_FSRDLY.COUNT</code> bit field is the number of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion.

Fault System Reset Delay Current Register

The SEC fault system reset delay current register (`SEC_FSRDLY_CUR`) contains the active count (`SEC_FSRDLY_CUR.COUNT` field) in (SEC) clock periods for the delay from fault active to system reset assertion, if enabled. The count is loaded from the `SEC_FSRDLY` register when a fault becomes active (`SEC_FSTAT.ACT` bit is set). The SEC decrements the value in `SEC_FSRDLY_CUR` each (SEC) clock cycle while the `SEC_FSTAT.ACT` bit is set.

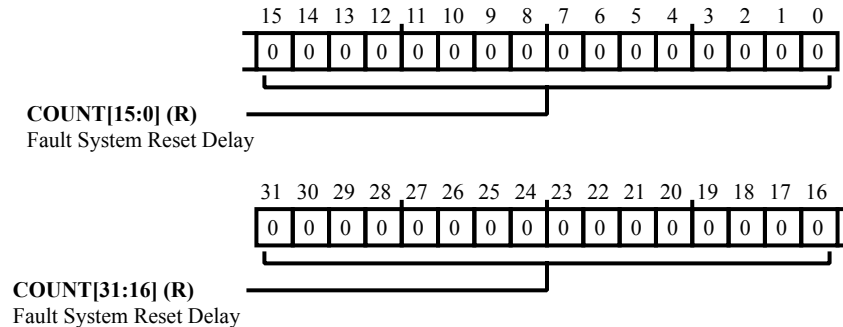


Figure 6-22: `SEC_FSRDLY_CUR` Register Diagram

Table 6-24: `SEC_FSRDLY_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault System Reset Delay. The <code>SEC_FSRDLY_CUR.COUNT</code> bit field is the active count in (SEC) clock periods for the delay from fault active to system reset assertion.

Fault Status Register

The SEC fault status register (`SEC_FSTAT`) indicates the operational status of the SFI.

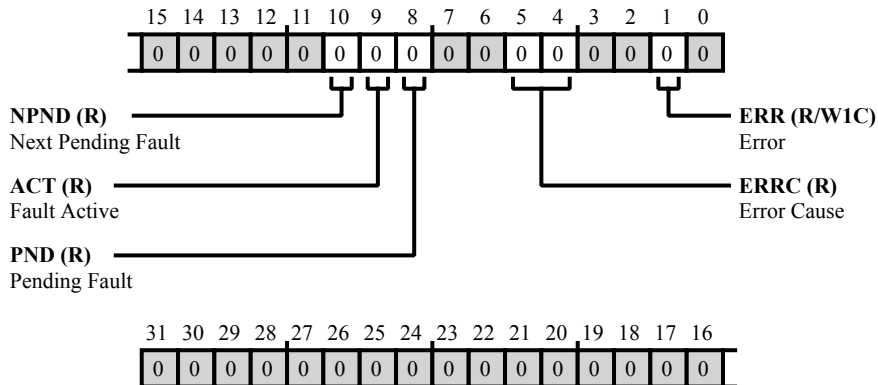


Figure 6-23: SEC_FSTAT Register Diagram

Table 6-25: SEC_FSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/NW)	NPND	Next Pending Fault. The <code>SEC_FSTAT.NPND</code> bit indicates that one or more sources have signaled fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the <code>SEC_FSTAT.NPND</code> bit when the fault interface detects assertion of any enabled fault source input, while either the <code>SEC_FSTAT.PND</code> or <code>SEC_FSTAT.ACT</code> bits are set. The SEC clears the <code>SEC_FSTAT.NPND</code> bit when there are no fault sources waiting.
		0 Not Pending
		1 Pending
9 (R/NW)	ACT	Fault Active. The <code>SEC_FSTAT.ACT</code> bit indicates that the SEC has received a fault source input, the current fault delay count (in the <code>SEC_FDLY_CUR</code> register) has expired, and the fault actions are enabled. The SEC also sets the <code>SEC_FSTAT.ACT</code> bit on fault input detection if the <code>SEC_FCTL.FIEN</code> bit is set. The <code>SEC_FSTAT.ACT</code> bit is cleared by writing the ID value of the asserted fault from <code>SEC_FSID</code> register to the <code>SEC_FEND</code> register.
		0 No Fault
		1 Active Fault

Table 6-25: SEC_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/NW)	PND	<p>Pending Fault.</p> <p>The SEC_FSTAT.PND bit indicates a fault source has signaled a fault assertion to the SEC, but the SEC has not yet triggered the event actions due to the delay selected with the SEC_FDLY register. The SEC fault interface sets the SEC_FSTAT.PND bit when the SEC_FSID is updated on assertion of a fault source input. The SEC_FSTAT.PND bit is only set when the SEC_FSTAT.ACT bit is cleared. The SEC updates the SEC_FSID register with the SID value when the SEC_FSTAT.PND bit is set. The SEC_FSTAT.PND bit is cleared <i>either</i> by the SEC fault interface when the current delay count in the SEC_FDLY_CUR register expires <i>or</i> by writing the SEC_FSID.SID field value (which indicates the ID of the asserted fault) to the SEC_FEND register.</p>
		0 Not Pending
		1 Pending
5:4 (R/NW)	ERRC	<p>Error Cause.</p> <p>When the SEC_FSTAT.ERR bit is asserted, the SEC updates SEC_FSTAT.ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. When the error type is end error, the status indicates that an end was received for a source while the SEC_FSTAT.ACT bit was not set.</p>
		0 Source Overflow Error
		1 Reserved
		2 End Error
		3 Reserved
1 (R/W1C)	ERR	<p>Error.</p> <p>The SEC_FSTAT.ERR bit indicates an SEC fault interface error. When SEC_FSTAT.ERR is set, the SEC updates the SEC_FSTAT.ERRC field to indicate the corresponding error cause. When multiple errors occur, the SEC_FSTAT register captures the status for the first error and does not capture subsequent errors until the status is cleared.</p>
		0 No Error
		1 Error Occurred

Global Control Register

The SEC global control register (`SEC_GCTL`) provides register locking, reset, and enable for the SEC module.

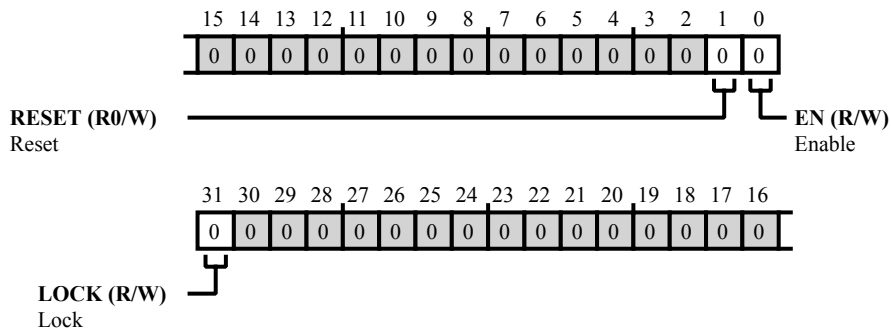


Figure 6-24: SEC_GCTL Register Diagram

Table 6-26: SEC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>SEC_GCTL.LOCK</code> bit is enabled, the <code>SEC_GCTL</code> register is read only.
		0 Unlock
		1 Lock
1 (R0/W)	RESET	Reset. The <code>SEC_GCTL.RESET</code> bit is write-1-action and triggers a soft reset to all SEC registers.
		0 No Action
		1 Reset
0 (R/W)	EN	Enable. The <code>SEC_GCTL.EN</code> bit is read/write and must be set for the SEC to begin/resume SEC operation with the current configuration and status. Clearing the <code>SEC_GCTL.EN</code> bit halts the execution of the SFI and all SCIs. All SSIs remain active, along with all error detection, without resetting status registers.
		0 Disable
		1 Enable

Global Status Register

The SEC global status register (`SEC_GSTAT`) contains global status bits for the SEC.

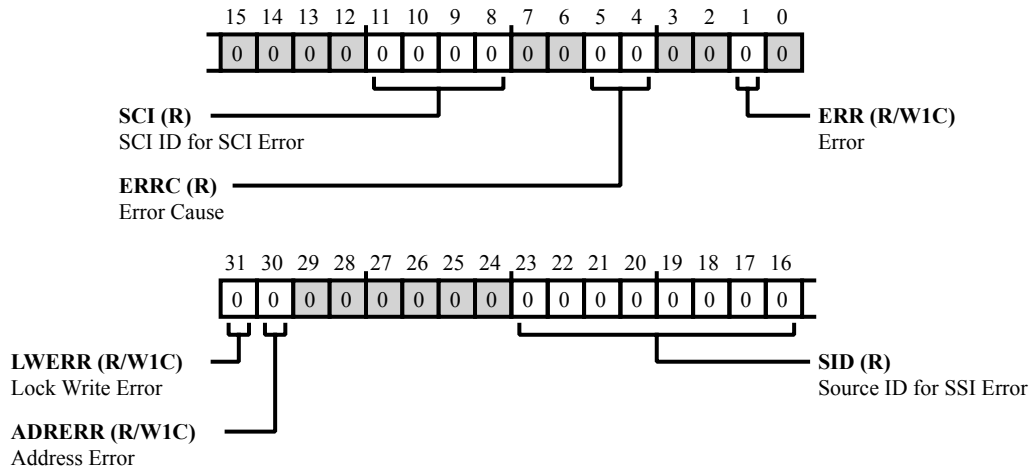


Figure 6-25: SEC_GSTAT Register Diagram

Table 6-27: SEC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	Lock Write Error. The <code>SEC_GSTAT.LWERR</code> bit indicates (when set) there was an attempted write to an SEC register while the <code>SEC_GCTL.LOCK</code> bit was set and while the global lock bit was enabled (<code>SPU_CTL.GLCK</code> bit =1). This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
30 (R/W1C)	ADRERR	Address Error. The <code>SEC_GSTAT.ADRERR</code> bit indicates that the SEC generated an address error. This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
23:16 (R/NW)	SID	Source ID for SSI Error. The <code>SEC_GSTAT.SID</code> bits indicate the source ID that generated the last SSI error conveyed in the <code>SEC_GSTAT.ERRC</code> field.
11:8 (R/NW)	SCI	SCI ID for SCI Error. The <code>SEC_GSTAT.SCI</code> bits indicate the number for the specific SCI that generated the last SCI error conveyed in the <code>SEC_GSTAT.ERRC</code> field.

Table 6-27: SEC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5:4 (R/NW)	ERRC	Error Cause. When the SEC updates the SEC_GSTAT.ERR bit, the SEC updates the SEC_GSTAT.ERRC bits to indicate the error type. Note that for SCI errors, the error status represents an OR of all the errors from each SCI. Note that for SSI errors, the error status indicates an error is active for any SSI input. This error is an OR of all the interrupt source errors.
		0 SFI Error
		1 SCI Error
		2 SSI Error
		3 Reserved
1 (R/W1C)	ERR	Error. The SEC_GSTAT.ERR bit indicates an error has occurred in the SEC. When the SEC asserts this bit (=1), the SEC updates the SEC_GSTAT.ERRC field to indicate the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of this bit. This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred

Global Raise Register

The SEC global raise register (`SEC_RAISE`) contains a source ID event set-to-pending field (`SEC_RAISE.SID`). When a source ID value is written to this field, the SEC raises the source's event status to pending.

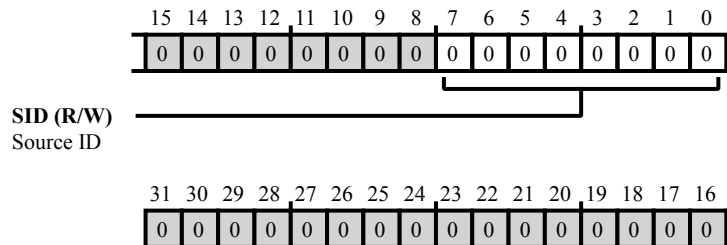


Figure 6-26: SEC_RAISE Register Diagram

Table 6-28: SEC_RAISE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID. The <code>SEC_RAISE.SID</code> bit field is the source ID of event that is set to pending status.

Source Control Register n

The SEC source control register (`SEC_SCTL[n]`) contains control bits to configure the SEC event sources. This register controls the configuration of the corresponding SEC event source.

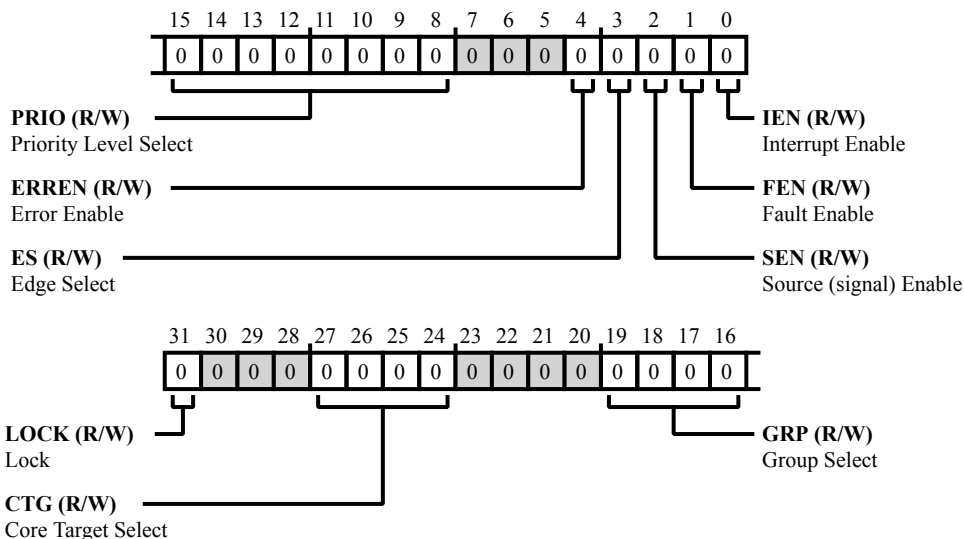


Figure 6-27: SEC_SCTL[n] Register Diagram

Table 6-29: SEC_SCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock is enabled (<code>SPU_CTL . GLCK</code> bit =1) and the <code>SEC_SCTL[n] . LOCK</code> bit is enabled, the <code>SEC_SCTL[n]</code> register is read only.
		0 Unlock
		1 Lock
27:24 (R/W)	CTG	Core Target Select. The <code>SEC_SCTL[n] . CTG</code> bits selects the specific SEC core interface to which the interrupt is mapped. Each system interrupt is mapped uniquely to one specific SEC core interface and (as a result) to a specific core.

Table 6-29: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration				
19:16 (R/W)	GRP	<p>Group Select.</p> <p>The SEC_SCTL[n].GRP bits each select a specific group for the interrupt. Each system interrupt can be assigned to any combination of groups supported by the SEC_SCTL[n].GRP field.</p> <p>For example, consider the situation where SEC_SCTL[n].GRP0 represents interrupt group 0, SEC_SCTL[n].GRP1 represents interrupt group 1, and so on. One group might be used for all enabled interrupts (for example, group 0) and an additional group might be used for all wakeup interrupts (for example, group 1). This approach supports a model of all interrupts and just the wakeup subset.</p> <p>Before going to idle or sleep, all non-wakeup interrupts can be masked off to allow only wakeup interrupts to be enabled for service. Selecting no group (all SEC_SCTL[n].GRP bits = 0) places the interrupt source in the category of "un-grouped".</p>				
15:8 (R/W)	PRIO	<p>Priority Level Select.</p> <p>The SEC_SCTL[n].PRIO bits sets the relative priority for an interrupt request. A pending interrupt request forwards its SEC_SCTL[n].PRIO value to the SEC core interface.</p>				
4 (R/W)	ERREN	<p>Error Enable.</p> <p>The SEC_SCTL[n].ERREN bit permits the SEC_SSTAT[n].ERR status bit to be set on error detection. If SEC_SCTL[n].ERREN is cleared, no errors are detected.</p> <table border="1" data-bbox="620 1199 1529 1297"> <tr> <td>0</td> <td>Disable</td> </tr> <tr> <td>1</td> <td>Enable</td> </tr> </table>	0	Disable	1	Enable
0	Disable					
1	Enable					
3 (R/W)	ES	<p>Edge Select.</p> <p>The SEC_SCTL[n].ES bit selects the operational and sensitivity mode of the SEC source interface input.</p> <table border="1" data-bbox="620 1430 1529 1528"> <tr> <td>0</td> <td>Level Sensitive</td> </tr> <tr> <td>1</td> <td>Edge Sensitive</td> </tr> </table>	0	Level Sensitive	1	Edge Sensitive
0	Level Sensitive					
1	Edge Sensitive					
2 (R/W)	SEN	<p>Source (signal) Enable.</p> <p>The SEC_SCTL[n].SEN bit controls whether the system event source input signal may affect the pending status of the source. Clearing the SEC_SCTL[n].SEN bit disables the source input signal from affecting the pending status. Setting SEC_SCTL[n].SEN enables the source input signal to affect the pending status.</p> <table border="1" data-bbox="620 1724 1529 1822"> <tr> <td>0</td> <td>Disable</td> </tr> <tr> <td>1</td> <td>Enable</td> </tr> </table>	0	Disable	1	Enable
0	Disable					
1	Enable					

Table 6-29: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	FEN	Fault Enable. The SEC_SCTL[n].FEN bit controls whether the SEC may forward an interrupt request to the SEC fault interface as a fault source. This bit does not affect the ability of an interrupt source to set an interrupt as pending. The SEC_SCTL[n].FEN bit only affects whether the pending request may be forwarded to the SEC fault interface.
		0 Disable
		1 Enable
0 (R/W)	IEN	Interrupt Enable. The SEC_SCTL[n].IEN bit controls whether the SEC may forward an interrupt request to a core for servicing. This bit does not affect the ability of an interrupt source to set an interrupt as pending.
		0 Disable
		1 Enable

Source Status Register n

The SEC event source status register (`SEC_SSTAT[n]`) contains bits indicating the status of the corresponding event source n. An event source may be: pending, active, active and pending, or neither pending nor active.

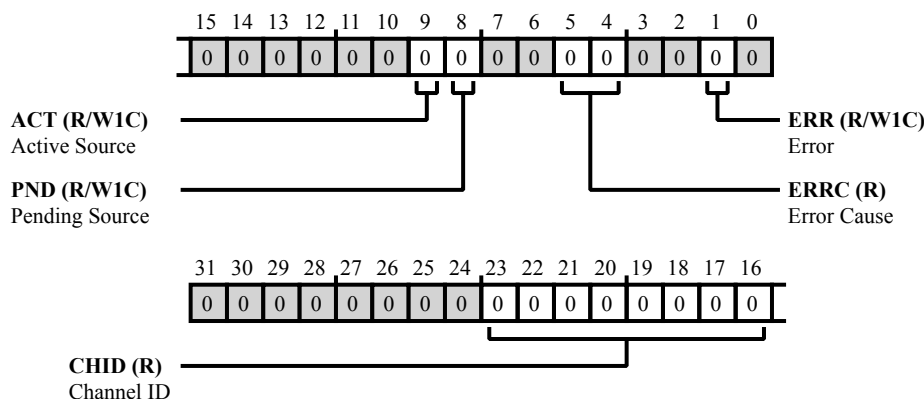


Figure 6-28: SEC_SSTAT[n] Register Diagram

Table 6-30: SEC_SSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	CHID	Channel ID. The <code>SEC_SSTAT[n].CHID</code> bits indicate the ID of the specific source (from a set of sources sharing one SEC source interface input) that asserted the SEC source interface input. An SEC source interface input may support multiple system sources, in which case the assertion must be qualified by an identifier to determine the channel that generated the assertion. The <code>SEC_SSTAT[n].CHID</code> field provides this value in the form of a numeric reference that is mapped to a specific interrupt source. The prioritization for simultaneously asserted sources is according to ID, with 0 being the highest priority. The <code>SEC_SSTAT[n].CHID</code> is captured when the SEC source interface input is acknowledged.
9 (R/W1C)	ACT	Active Source. The <code>SEC_SSTAT[n].ACT</code> bit indicates the source has been accepted by a core for servicing, but the service is not yet complete. An <code>SEC_SSTAT[n].ACT</code> bit is set by the SEC when the specific system interrupt is acknowledged by the core through the SEC core interface. An <code>SEC_SSTAT[n].ACT</code> bit is cleared by the SEC when the core provides interrupt service end indication for the specific system interrupt through the SEC core interface.
		0 Not Active
		1 Active

Table 6-30: SEC_SSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	PND	<p>Pending Source.</p> <p>The SEC_SSTAT[n].PND bit indicates the source has signaled an event request, but the event request has not been (or is not currently being) serviced. A SEC_SSTAT[n].PND bit is set by the SEC on detection of an assertion of the corresponding system source input. A SEC_SSTAT[n].PND bit is cleared by the SEC when the specific system event is acknowledged by the core through the SEC core interface or by a W1C operation.</p>
		0 Not Pending
		1 Pending
5:4 (R/NW)	ERRC	<p>Error Cause.</p> <p>When the SEC_SSTAT[n].ERR bit is asserted, the SEC updates SEC_SSTAT[n].ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. When the error type is end error, the status indicates that an end was received for a source while the SEC_SSTAT[n].ACT bit was not set.</p>
		0 Source Overflow Error
		1 Reserved
		2 End Error
		3 Reserved
1 (R/W1C)	ERR	<p>Error.</p> <p>The SEC_SSTAT[n].ERR bit indicates an error for a specific system interrupt source. When the SEC_SSTAT[n].ERR bit is set, the SEC updates the SEC_SSTAT[n].ERRC field to the value of the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of the SEC_SSTAT[n].ERR bit.</p>
		0 No Error
		1 Error Occurred

7 Trigger Routing Unit (TRU)

The TRU provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering
- Synchronization of concurrent activities

TRU Features

The TRU supports the following features:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes. Once a DMA channel completes data transfer, it can act as a Trigger Master and signal an internal trigger pulse to the programmed Trigger Slave which can also be another DMA channel. The Slave Trigger connected to the DMA channel kicks off the DMA transfer automatically. None of this requires core intervention once the initialization is done.
- Software triggers. The best use of triggers is to minimize core intervention. It is also possible to initiate a trigger pulse to a Trigger Slave, in the software.
- Synchronization of concurrent activities. A single Trigger Master can initiate a trigger pulse to multiple Trigger Slaves so that several system level activities can be synchronized on an internally or externally generated event.
- Configuration protection through register-level lock bits and global lock indication

TRU Functional Description

The following sections provide a description of the TRU.

ADSP-2156x TRU Register List

The Trigger Routing Unit (TRU) provides simple sequence control of distributed modules without the penalties associated with core intervention (for example, interrupt overhead). The TRU receives trigger inputs from all master trigger inputs (MTI) and the TRU master trigger register ([TRU_MTR](#)). Based on these inputs, the TRU logic generates trigger outputs that initiate slave operations in the processor core and peripherals. A set of registers governs TRU operations. For more information on TRU functionality, see the TRU register descriptions.

Table 7-1: ADSP-2156x TRU Register List

Name	Description
TRU_ERRADDR	Error Address Register
TRU_GCTL	Global Control Register
TRU_MTR	Master Trigger Register
TRU_SSR[n]	Slave Select Register
TRU_STAT	Status Information Register

ADSP-2156x TRU Interrupt List

Table 7-2: ADSP-2156x TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
112	TRU0_SLV0	TRU0 Interrupt 4 -- Core 0	Edge	
113	TRU0_SLV1	TRU0 Interrupt 5 -- Core 0	Edge	
114	TRU0_SLV2	TRU0 Interrupt 6 -- Core 0	Edge	
115	TRU0_SLV3	TRU0 Interrupt 7 -- Core 0	Edge	

ADSP-2156x TRU Trigger List

Table 7-3: ADSP-2156x TRU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 7-4: ADSP-2156x TRU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
78	TRU0_SLV0	TRU0 Interrupt Request 0	Pulse
79	TRU0_SLV1	TRU0 Interrupt Request 1	Pulse
80	TRU0_SLV2	TRU0 Interrupt Request 2	Pulse

Table 7-4: ADSP-2156x TRU Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
81	TRU0_SLV3	TRU0 Interrupt Request 3	Pulse

TRU Definitions

The following definitions are helpful when using the TRU module.

Trigger Master

A trigger master is any system module that provides trigger event indication to the TRU. Trigger master modules define trigger events and conditions for assertion.

Trigger Master ID

Trigger masters are assigned a unique numeric ID according to their physical connection to the TRU. Trigger master ID 0 is reserved and defined as null.

Trigger Slave

A trigger slave is any system module that receives a trigger event indication from the TRU. Trigger slave modules define a trigger event response.

NOTE: For peripherals configured as a slave receiver: if the corresponding peripheral DMA is kept waiting for trigger from a master, it can receive stale or garbage data. The peripheral receiver begins shifting in data as soon as it is enabled and it does not wait for the trigger master. Upon receiving the trigger, this data is moved into the DMA FIFO. Therefore, this sequence can result in the slave receiver getting unusable data.

TRU Block Diagram

Trigger masters and the Master Trigger Register (MTR) generate trigger assertions. Each trigger slave has a dedicated Slave Select Register (SSR) that specifies the unique trigger master from which it receives the trigger indication.

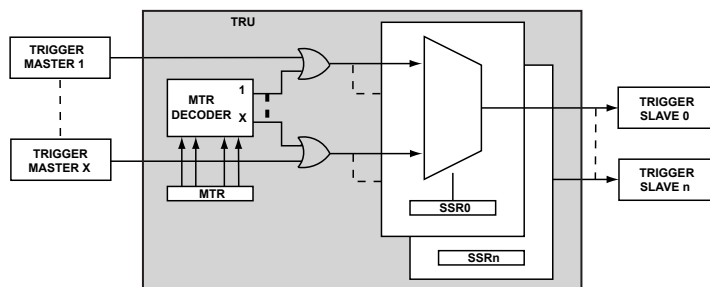


Figure 7-1: TRU Block Diagram

TRU Architectural Concepts

The TRU supports a simple trigger-in/trigger-out model for modules that comply with the triggering functional model. The TRU is the controller of the trigger system. Trigger outputs from trigger masters are mapped to trigger inputs of trigger slaves through a set of programmable registers (`TRU_SSR[n]`).

System modules are trigger master only, trigger slave only, or trigger master and trigger slave.

All of the trigger input and output signals are connected to a trigger routing unit (TRU) which manages the connections of triggers between modules.

In multi-processor systems, multiple TRU units are provided. These TRUs are networked together. Generic Trigger Ports (GTPs) are provided to forward trigger events from one TRU unit to another, forming a pathway from trigger masters to trigger slaves wherever they might lie in the system.

TRU Programming Model

Implementing sequence control using the TRU requires, at a minimum, proper configuration of a trigger slave, a trigger master, and the TRU module itself. The only requirement for the configuration procedure is that the trigger master is configured and enabled as the last step.

Complete the following other steps:

- Configure the trigger slave for response to triggers.
- Configure the TRU to map the trigger master to the trigger slave through the `TRU_SSR[n]` registers.
- Configure the trigger master to generate trigger assertions.
- Alternatively, use software triggering for trigger assertion. Writing the trigger master ID to the MTR register generates software triggers.

Programming Concepts

The following concepts aid in programming the TRU.

- **Trigger Sequence Configuration.** A simple sequence consists of one trigger master and one trigger slave. More complex trigger sequences consist of several trigger slaves functioning as trigger slave and trigger master. Additionally, trigger sequences can loopback to the original master forming a perpetual sequence.
- **Software Triggering.** Writing a trigger master ID to the MTR generates a trigger within the TRU from the trigger master ID specified.
- **Synchronization.** The TRU can be used to coarsely synchronize events by mapping multiple trigger slaves to the same trigger master or by generating multiple trigger master assertions simultaneously through the MTR.
- **Configuration Protection.** The `TRU_SSR[n].LOCK` bit and the `TRU_GCTL.LOCK` bit enable register level write-protection when the global lock is asserted in the SPU.

Programming Examples

The following examples show the steps to create a single trigger.

Configuring a Simple Trigger Sequence

The following example shows the steps to create a simple trigger.

1. Enable the Global Lock feature bit on the SPU (set the `SPU_CTL.GLCK` bit).
2. Set the `TRU_GCTL.LOCK` bit to 1 so that the `TRU_GCTL` register becomes read-only.
3. Set the `TRU_GCTL.MTRL` bit to 1 so that the `TRU_MTR` register becomes read-only.

TRU Event Control

The TRU is a major part of event control solutions. It is the center of the trigger functional model and can extend to support the interrupt and fault management models as well.

TRU Status and Error Signals

The TRU does not have dedicated status and error output signals other than the MMR interface. Slave errors are reported to the master over the standard peripheral bus protocol.

ADSP-2156x TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 7-5: ADSP-2156x TRU Register List

Name	Description
<code>TRU_ERRADDR</code>	Error Address Register
<code>TRU_GCTL</code>	Global Control Register
<code>TRU_MTR</code>	Master Trigger Register
<code>TRU_SSR[n]</code>	Slave Select Register
<code>TRU_STAT</code>	Status Information Register

Error Address Register

The TRU error address register ([TRU_ERRADDR](#)) holds the address from the memory-mapped register access generating an access error of TRU registers.

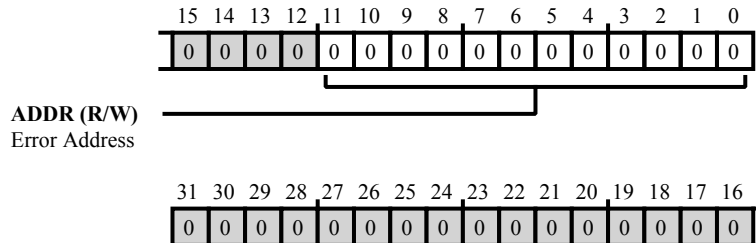


Figure 7-2: TRU_ERRADDR Register Diagram

Table 7-6: TRU_ERRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ADDR	<p>Error Address.</p> <p>The <code>TRU_ERRADDR.ADDR</code> holds the address from the memory-mapped register access generating an access error of TRU registers. These errors occur on access to the TRU_SSR[n] or TRU_MTR registers when these registers are locked or on access to an invalid address. See the TRU_SSR[n] and TRU_MTR register descriptions for more information about locking.</p> <p>The TRU_ERRADDR register holds the address of the first error to occur. In the event of multiple errors occurring, the TRU_ERRADDR register contains the address of the first error. To re-enable the TRU_ERRADDR register for update, both status bits (<code>TRU_STAT.LWERR</code> and <code>TRU_STAT.ADDRERR</code>) in the TRU_STAT register must be cleared.</p>

Global Control Register

The TRU global control register ([TRU_GCTL](#)) provides register locking, TRU reset, and TRU enable.

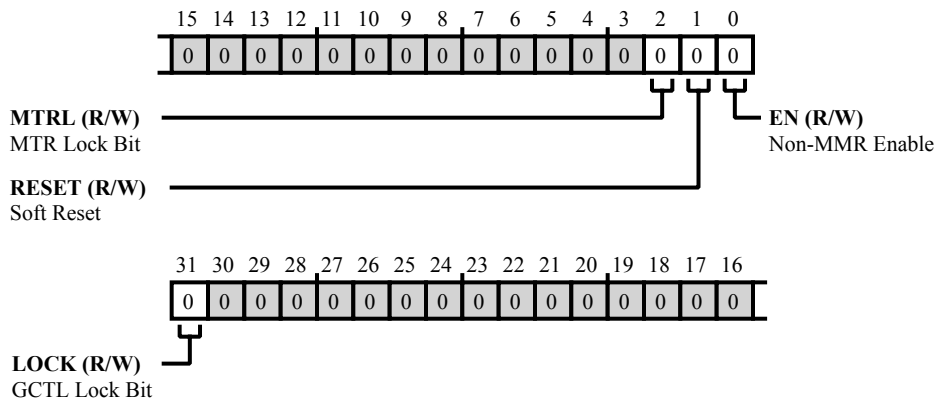


Figure 7-3: TRU_GCTL Register Diagram

Table 7-7: TRU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	GCTL Lock Bit. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>TRU_GCTL.LOCK</code> bit is enabled, the <code>TRU_GCTL</code> register is read only.
		0 Read write
		1 Read only
2 (R/W)	MTRL	MTR Lock Bit. If the global lock is enabled (<code>SPU_CTL.GLCK</code> bit =1) and the <code>TRU_GCTL.MTRL</code> bit is enabled, the <code>TRU_MTR</code> register is read only.
		0 Read write
		1 Read only
1 (R/W)	RESET	Soft Reset. The <code>TRU_GCTL.RESET</code> bit is write-1-action and triggers a soft reset to all TRU registers.
		0 No action
		1 Soft reset

Table 7-7: TRU_GCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	EN	<p>Non-MMR Enable.</p> <p>The TRU_GCTL.EN bit is read/write and must be set for the TRU to propagate trigger events. All TRU register read/write operations continue to operate independent of the TRU_GCTL.EN bit.</p>
		0 No trigger events
		1 Propagate trigger events

Master Trigger Register

The TRU master trigger register (`TRU_MTR`) permits trigger generation through software by writing a trigger master ID value to one of the four fields in the `TRU_MTR` register. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `TRU_GCTL.LOCK` bit is set, the `TRU_MTR` register is read only. Note this register is primarily used for debug to trigger a TRU output

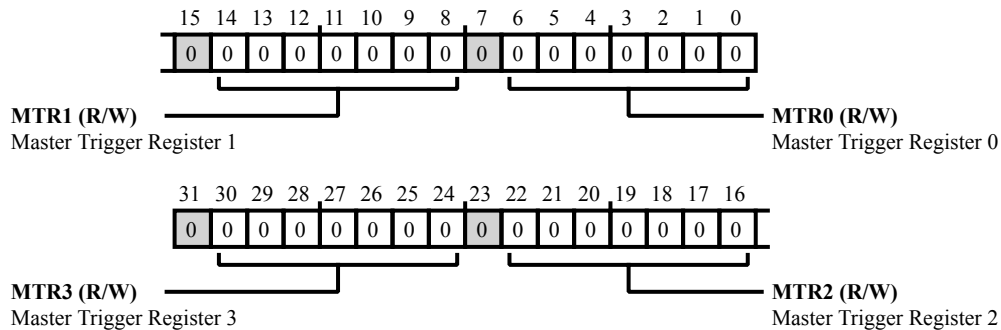


Figure 7-4: TRU_MTR Register Diagram

Table 7-8: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:24 (R/W)	MTR3	Master Trigger Register 3. The <code>TRU_MTR.MTR3</code> bit field is the trigger master ID value for master 3.
		0 No master specified
		1-105 Range of valid masters
22:16 (R/W)	MTR2	Master Trigger Register 2. The <code>TRU_MTR.MTR2</code> bit field is the trigger master ID value for master 2.
		0 No master specified
		1-105 Range of valid masters
14:8 (R/W)	MTR1	Master Trigger Register 1. The <code>TRU_MTR.MTR1</code> bit field is the trigger master ID value for master 1.
		0 No master specified
		1-105 Range of valid masters
6:0 (R/W)	MTR0	Master Trigger Register 0. The <code>TRU_MTR.MTR0</code> bit field is the trigger master ID value for master 0.
		0 No master specified
		1-105 Range of valid masters

Slave Select Register

The TRU slave select registers (`TRU_SSR[n]`) each provide slave selection and register locking.

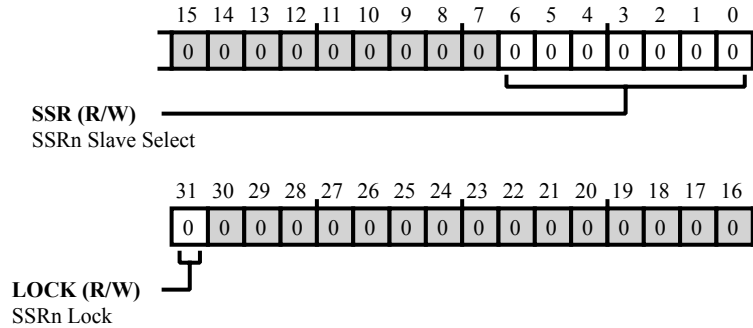


Figure 7-5: TRU_SSR[n] Register Diagram

Table 7-9: TRU_SSR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	SSRn Lock. If the global lock is enabled (<code>SPU_CTL . GLCK</code> bit =1) and the <code>TRU_SSR[n] . LOCK</code> bit is enabled, the <code>TRU_SSR[n]</code> register is read only.
		0 Unlock register
		1 Lock register
6:0 (R/W)	SSR	SSRn Slave Select. The <code>TRU_SSR[n]</code> register selects the trigger master ID to which the trigger slave responds. For example, when a <code>TRU_SSR[n]</code> register is set to respond to trigger master ID n, a trigger that is generated by trigger master ID n results in a trigger out to the slave.
		0 No master specified
		1-105 Range of valid masters

Status Information Register

The TRU status register (`TRU_STAT`) contains the status of `TRU_MTR` and `TRU_SSR[n]` register writes and status of bus read/write errors.

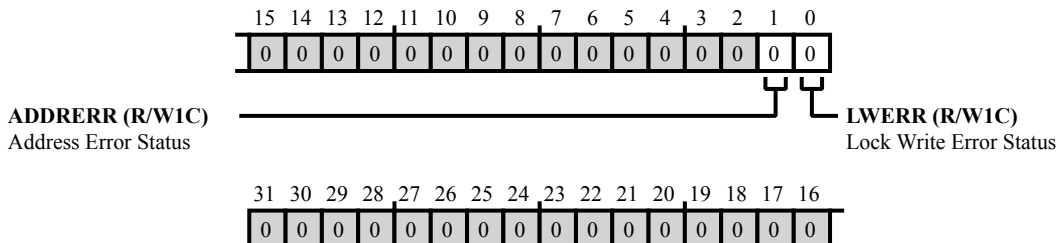


Figure 7-6: TRU_STAT Register Diagram

Table 7-10: TRU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	ADDRERR	Address Error Status. The <code>TRU_STAT.ADDRERR</code> bit is set when an invalid address is provided for an MMR access while the TRU is selected. Writing a one to this bit clears the error indication. The <code>TRU_ERRADDR</code> register also is updated when an address error occurs during an MMR access while the TRU is selected.
		0 No error
		1 Error occurred
0 (R/W1C)	LWERR	Lock Write Error Status. If <code>TRU_STAT.LWERR</code> is set, a lock write error has occurred. Writing a one to this bit clears the error indication.
		0 No error
		1 Error occurred

8 L2 System Memory

L2 system memories have significant bandwidth for core accesses, but it is important to note that L2 responds slower to core accesses than L1 memories. L2 SRAM is the ideal storage for multiple processor cores to share data and instruction resources, such as semaphores, shared buffers, and code libraries. Due to sophisticated data integrity protection and write protection, L2 SRAM is also ideal for data and instructions critical for safe operation of the application.

L2 System Memory Features

The L2 system memory features include:

- Operation at SYSCLK frequency
- ECC protection of SRAM area
- ECC memory refresh

There is one instance of L2 system memory: L2CTL0.

- L2CTL0 contains 1M byte of RAM grouped into eight banks, 128K bytes each and 96K bytes of boot ROM.

L2 System Memory Functional Description

The L2 system memory manages all of the L2 SRAM and ROM memory banks. The system memory interface arbitrates competing accesses, write protection, and ensures SRAM data integrity. The L2 system memory domain is a unified instruction and data memory. It can hold any mixture of code and data required by the system design.

The following sections provide a functional description of the L2 system memory.

ADSP-2156x L2CTL Register List

Table 8-1: ADSP-2156x L2CTL Register List

Name	Description
L2CTL_CTL	Control Register
L2CTL_EADDR0	Error Type 0 Address Register

Table 8-1: ADSP-2156x L2CTL Register List (Continued)

Name	Description
L2CTL_EADDR1	Error Type 1 Address Register
L2CTL_ERRADDR0	ECC Error Address 0 Register
L2CTL_ERRADDR1	ECC Error Address 1 Register
L2CTL_ERRADDR2	ECC Error Address 2 Register
L2CTL_ERRADDR3	ECC Error Address 3 Register
L2CTL_ERRADDR4	ECC Error Address 4 Register
L2CTL_ERRADDR5	ECC Error Address 5 Register
L2CTL_ERRADDR6	ECC Error Address 6 Register
L2CTL_ERRADDR7	ECC Error Address 7 Register
L2CTL_ERRADDR8	ECC Error Address 8 Register
L2CTL_ET0	Error Type 0 Register
L2CTL_ET1	Error Type 1 Register
L2CTL_INIT	Initialization Register
L2CTL_ISTAT	Initialization Status Register
L2CTL_PCTL	Power Control Register
L2CTL_REVID	Revision ID Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_SADR	Scrub Start Address Register
L2CTL_SCNT	Scrub Count Register
L2CTL_SCTL	Scrub Control Register
L2CTL_STAT	Status Register
L2CTL_WPCR	Write Priority Count Register

ADSP-2156x L2CTL Interrupt List

Table 8-2: ADSP-2156x L2CTL Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
9	L2CTL0_ECC_ERR	L2CTL0 ECC Error	Level	
10	Reserved	Reserved	Reserved	Reserved
11	L2CTL0_EVT	L2CTL0 Scrub/Initialization Done	Level	

ADSP-2156x L2CTL Trigger List

Table 8-3: ADSP-2156x L2CTL Trigger List Masters

Trigger ID	Name	Description	Sensitivity
94	L2CTL0_EVT	L2CTL0 L2 Memory Event	Level

Table 8-4: ADSP-2156x L2CTL Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

L2 System Memory Block Diagram

The *L2 System Block Diagram* shows the complete L2 system memory, including the memory block instance L2CTL0. The L2CTL0 block contains boot ROM code and three banks of L2 ROM containing 96 Kbytes each.

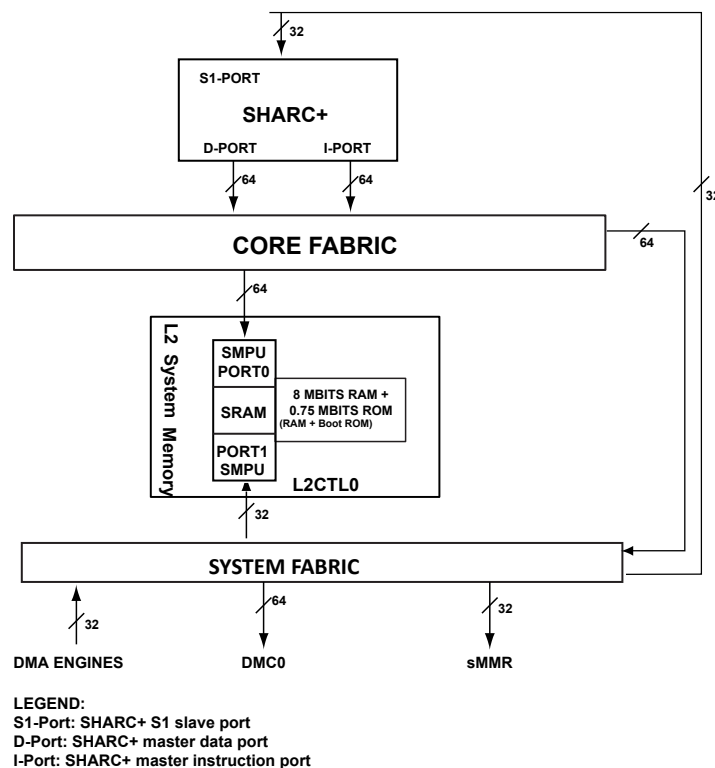


Figure 8-1: L2 System Block Diagram

L2 System Memory Architectural Concepts

The following sections describe architecture features of the L2 system memory.

- [Read/Write Latency and Throughput](#)
- [Arbitration and Priority](#)

Access Characteristics

The L2 system memory interface converts all 8-bit, 16-bit, and 32-bit accesses to 64-bit accesses. Additionally, it converts 8-bit, 16-bit, and 32-bit bursts to an equivalent internal 64-bit access. For example, the L2 system memory interface converts a 64-bit address-aligned burst of 8-bit accesses of burst length 8 to a single 64-bit access.

Read/Write Latency and Throughput

The L2 memory design is optimized for burst accesses at the crossbar interface. The L2 system memory buffers and converts write data of 8/16/32-bit to an equivalent 64-bit access. This conversion creates modulo-32-bit writes if the starting addresses are 32-bit aligned. A single 8-bit or 16-bit access, or a non-32-bit address-aligned 8-bit or 16-bit burst access to an ECC-enabled bank creates an extra latency of two SYSCLK cycles. No extra latency is seen if the ECC is disabled.

NOTE: Continuous 8/16-bit core access to an ECC-enabled L2 bank is not recommended from a throughput perspective.

L2 Memory Controller Block Diagram (Instance)

As shown in the *L2 System Memory Block Diagram*, the L2 controller has two ports that interface to system crossbars. Port 0 is a 64-bit interface that is dedicated to core and MDMA3 traffic, and port 1 is a 32-bit interface that connects through the rest of the DMA access. For L2 SRAM both ports (0/1) have a read channel and a write channel, for L2 ROM both ports (0/1) have read channels only. The SRAM is organized in multiple banks, and each bank has 128K Bytes. The 96KB ROM is divided into three banks of 32KB.

Within each bank, data is organized into 16384 words, with each word comprising 64 bits of data and 14 bits of ECC checksum. ROM memory is not protected by the ECC scheme. When the L2 controller accesses RAM and ROM cells, it always reads and writes whole 64-bit words. Despite this, the L2 controller supports 8-, 16-, and 32-bit reads and writes from cores and system by applying respective data masks.

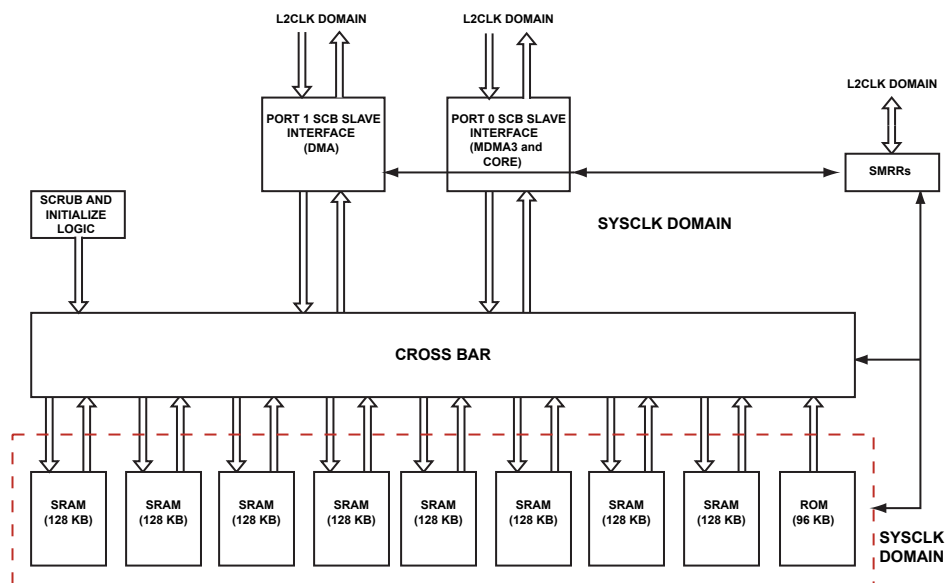


Figure 8-2: L2 System Memory Block Diagram

Arbitration and Priority

Each bank of L2 RAM or ROM has an arbiter which receives requests from the two crossbar ports.

Each arbiter follows a fixed priority scheme for giving grants when more than one channel requests the same bank. The arbiter also supports priority elevation through urgent priority requests.

NOTE: Attempting a write access to both L2 ROM spaces returns an error.

The *Fixed Priority* table shows the priority for fixed priority mode (with urgent priority disabled) for each SCB channel.

If two cores (or the 64-bit Max BW DMA) simultaneously try to access L2 for the same instance (both read or both write), even to different banks, software allows only one master access at a time. One access port can support one read and write at the same time. However, if one core issues a write and the other issues a read, then access can proceed simultaneously. There is no extra latency inside L2, as long as the accesses are to different banks (assuming pending DMA traffic is also to a non-conflicting bank).

When a core and DMA both access the same bank via the same port (both read or both writes), the best access rate that DMA can achieve is one 64-bit access in every three SCLK cycles during the conflict period. This access rate is achieved by programming the read priority count register (`L2CTL_RPCR.RPC0`) bit and the write priority count register (`L2CTL_WPCR.WPC0`) bit to 0, while programming the `L2CTL_RPCR.RPC1` and the `L2CTL_WPCR.WPC1` bits to 1.

The arbiters also support priority elevation for a particular channel that has been starved of grants for many SCLK cycles. If a channel does not get a grant for N cycles after its request, then that channel can elevate the priority of its request by issuing an urgent priority request. This request causes that particular channel to become the highest priority master for the next grant cycle (pipelined arbitration for urgent priority). The number of cycles N , after which the priority is elevated, can be programmed for each channel separately using the `L2CTL_RPCR` and `L2CTL_WPCR` registers.

Programming the bits in the `L2CTL_RPCR` and `L2CTL_WPCR` registers appropriately achieves the best grant rate for DMA. This grant rate of one in three SCLK cycles during the conflict period is achievable under the following conflict conditions:

- An access conflict between the core and DMA to the same memory bank in the fixed priority arbitration scheme with core activity always prioritized over DMA activity
- An access conflict within the pipelined implementation of urgent priority

To disable urgent priority requests, set the `L2CTL_CTL.DISURP` bit. This bit disables the urgent priority requests for all port channels. Each channel can also be prevented from raising the urgent priority request through the priority count register for the specific channel. However, there is no support for disabling urgent priority for a specific memory bank arbiter.

The *Fixed Priority With Priority Elevation* table provides the various priority levels for the L2 system memory.

Table 8-5: Fixed Priority With Priority Elevation

Channel	Priority Level
L2 Refresh Request	9 (highest)
Port 0 Read Channel Urgent Request	8
Port 0 Write Channel Urgent Request	7
Port 1 Read Channel Urgent Request	6
Port 1 Write Channel Urgent Request	5
Port 0 Read Channel Normal Request	4
Port 0 Write Channel Normal Request	3
Port 1 Read Channel Normal Request	2
Port 1 Write Channel Normal Request	1 (lowest)

Data Integrity

The following sections provide information on how the L2 system memory ensures data integrity.

ECC Algorithm

Hsaio encoding calculates the ECC syndrome. A 7-bit syndrome is generated during write operation and stored as a 7-bit parity field along with the 32 data bits. Each data bit contributes to three parity bits according. Each parity bit represents the XOR value of 13 or 14 data bits according to the following mapping:

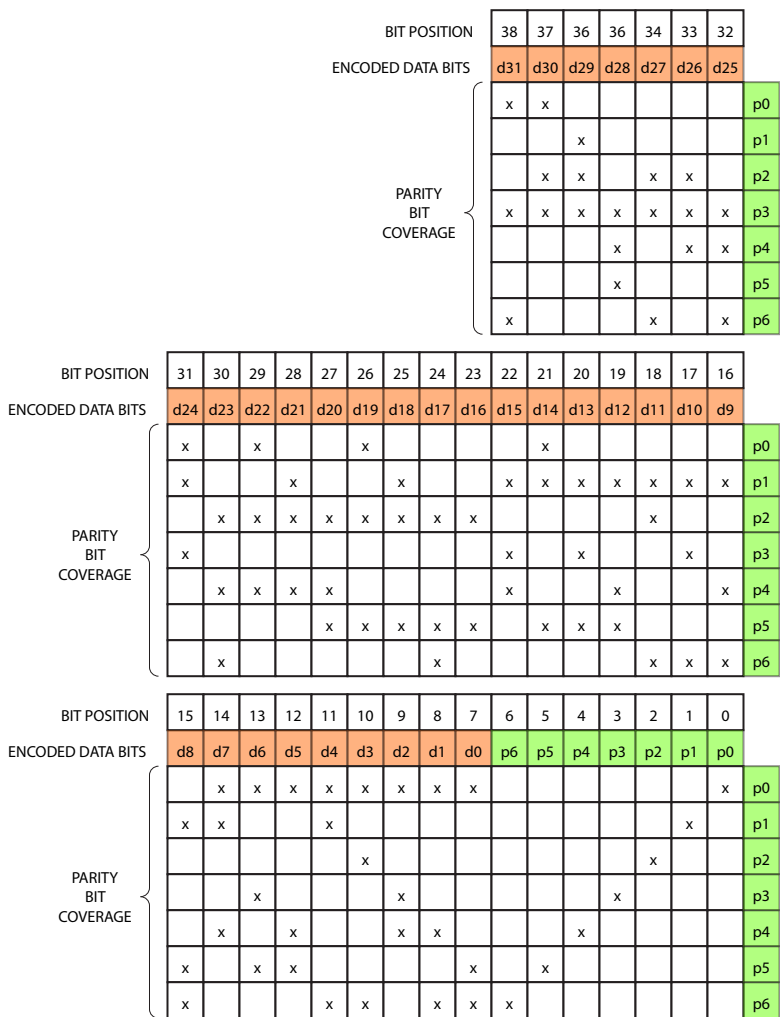


Figure 8-3: Hsaio Parity Bit Mapping

During read operation, the parity bits become part of the syndrome equation. The new syndrome bits are now the XOR values of the 13 or 14 data bits plus the respective stored parity bit. If any of the seven syndrome bits is set, an error situation is detected. An OR gate cross of the 7 bits reports the error, without specifying the type of the error.

If a single parity bit failed, the new 7-bit syndrome has 1 bit that is set. If a single data bit failed, the new syndrome has 3 bits that are set, because all three related parity bits fail. So, an XOR gate cross of all seven syndrome bits detects a single-bit error, indicating that an odd number of syndrome bits is set.

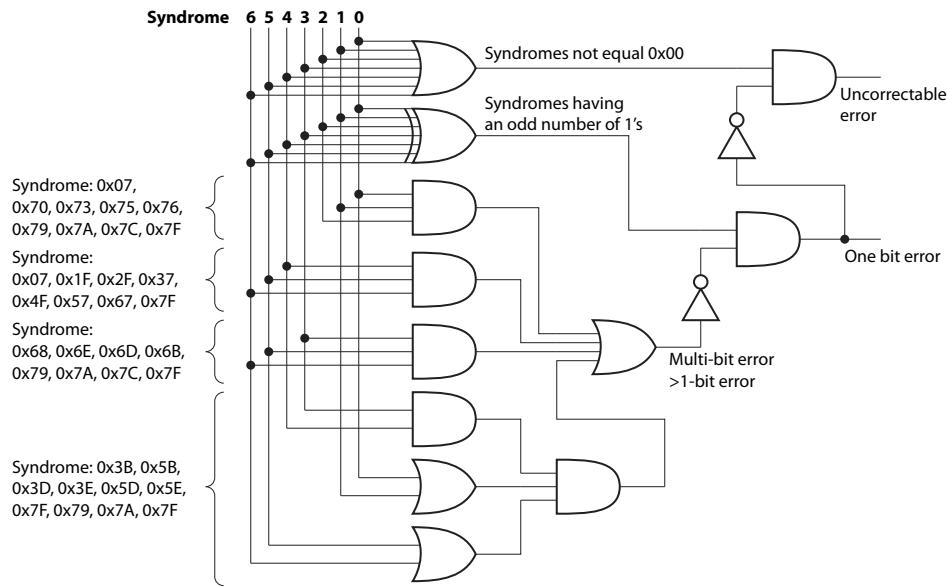


Figure 8-4: Hsiao Error Reports

The XOR gate detects single-bit errors and does not flag any dual-bit error. But, the gate does flag 50% of the other multi-bit errors undesirably. Extra logic is implemented to increase the detection rate of multi-bit errors to 68.7% as shown in the figure.

If a single-bit error is detected, the failing bit can be localized and corrected. If all three syndrome bits corresponding to a specific data bit are 1, a data error is assumed. The respective data bit is toggled on its way to the system bus.

ECC Hardware Control

After reset, ECC protection is enabled. The boot code initializes all L2 SRAM data and checksum cells. ECC protection adds some cycle penalty when 8-bit and 16-bit values write L2 memory. Disable ECC protection for individual SRAM banks by setting the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits. Due to caching mechanisms of the processor cores and data bursting of the DMA channels, 8-bit and 16-bit write accesses are rather uncommon. Typically, only two-dimensional DMA operations or uncached 8-bit and 16-bit store instruction can trigger these writes.

ECC Error Management

The L2 system memory flags 2-bit and multi-bit errors to the system by:

- Raising the `ECC_ERR` interrupt
- Reporting a read error to the system bus
- Setting the sticky `L2CTL_STAT.ECCERR7-L2CTL_STAT.ECCERR0` status flag
- Latching the address of the failing operation into the respective `L2CTL_ERRADDR7-L2CTL_ERRADDR0` register.

There is one error status bit and one error address register per L2 SRAM bank.

Typically, ECC_ERR events are declared as system faults in the system event controller (SEC). Whether these faults are reported, the interrupt service routine can consult the L2CTL_STAT register and the L2CTL_ERRADDR0 through L2CTL_ERRADDR7 registers to determine whether:

- The data at the failing L2 address was critical enough to require an immediate reboot of the system
- The data at the failing L2 address was less critical or can be restored

The L2CTL_STAT.ECCERR0 through L2CTL_STAT.ECCERR7 flags are cleared with a W1C operation.

Memory Refresh

If data in L2 SRAM contains single-bit errors, the data is corrected on its way to the system buses. The corrected value is not written back to the SRAM location. To prevent any risk of accumulation of single-bit errors over time and to minimize likelihood of multi-bit errors, the L2 system memory provides a special memory refresh mechanism.

If there are dual-bit or multi-bit errors, the ECC_ERR interrupt is raised, and data is not written back to memory.

Program the memory refresh or scrub using:

- The L2CTL_SADR register with the start address of the scrub
- The L2CTL_SCNT register with the total number of 64-bit addresses to be scrubbed starting from the L2CTL_SADR address

Program the number of cycles between each scrub using the L2CTL_SCTL register. During the scrub, the L2 system memory issues a read, followed by a write-back operation if there is a single-bit ECC error. Once the L2 system memory completes scrubbing the programmed memory region, it generates an interrupt and starts again from the L2CTL_SADR address unless the L2CTL_SCTL.SEN bit is disabled. If the L2CTL_SCTL.SEN bit is cleared before completing the programmed address range, the scrub stops after completing any already issued scrub access. The scrub read/writes always start from the full 64-bit equivalent of the address written into the L2CTL_SADR register. A 64-bit value is always read, and the 64-bit value is written back. The scrub access has the highest priority. Programs can configure 8-bit, 16-bit, or 32-bit addresses in the L2CTL_SADR register. The lower 3 bits of this register are treated as *do-not-care* values because the internal memory array is always accessed when using 64 bits.

Memory refresh operation is meaningless when the L2CTL_CTL.BK0EDIS through L2CTL_CTL.BK7EDIS disable bits are set.

Power Modes

Each L2 memory bank supports low-power modes.

If the bank is not in use, put each L2 system memory bank into the following low-power modes to save power.

Deep Sleep Mode

Set the L2CTL_PCTL.BK7DS through L2CTL_PCTL.BK0DS control bits to enter this mode. This mode preserves the memory contents.

Shut Down Mode

Set the `L2CTL_PCTL.BK7SD` through `L2CTL_PCTL.BK0SD` control bits to enter this mode. The memory banks do not retain the data (existing data is lost).

NOTE: It may take up to 11 SYSCLK cycles to deactivate the power mode after writing to the `L2CTL_PCTL` register. Therefore, L2 access must be issued after this delay.

Access to a memory bank in shut down or deep sleep mode may result in an unpredictable behavior. Ensure that initialization/refresh and memory access to such banks are not issued.

L2 System Memory Event Control

The following sections describe event control features of the L2 system memory, such as error response.

ECC Error Interrupt

A bus error is signaled under any of the following conditions.

- A write access to ROM address space
- A read/write access to reserved address space
- An ECC multi-bit error in an ECC-enabled bank. A non-modulo, 32-bit write to an ECC-enabled bank can also potentially create a bus error response due to an ECC multi-bit error. This response is because the L2 system memory implements a 32-bit ECC, and therefore a non-modulo, 32-bit write results in a read. This read can create multi-bit errors even if the memory was initialized.

Bus error notifications are stored in the `L2CTL_STAT` register, and the addresses that generated the error on a given port are stored in the `L2CTL_EADDR0/L2CTL_EADDR1` register of that port. The details of the error are stored in the `L2CTL_ET0/L2CTL_ET1` register of the port.

ADSP-2156x L2CTL Register Descriptions

Some description. (L2CTL) contains the following registers.

Table 8-6: ADSP-2156x L2CTL Register List

Name	Description
<code>L2CTL_CTL</code>	Control Register
<code>L2CTL_EADDR0</code>	Error Type 0 Address Register
<code>L2CTL_EADDR1</code>	Error Type 1 Address Register
<code>L2CTL_ERRADDR0</code>	ECC Error Address 0 Register
<code>L2CTL_ERRADDR1</code>	ECC Error Address 1 Register
<code>L2CTL_ERRADDR2</code>	ECC Error Address 2 Register

Table 8-6: ADSP-2156x L2CTL Register List (Continued)

Name	Description
L2CTL_ERRADDR3	ECC Error Address 3 Register
L2CTL_ERRADDR4	ECC Error Address 4 Register
L2CTL_ERRADDR5	ECC Error Address 5 Register
L2CTL_ERRADDR6	ECC Error Address 6 Register
L2CTL_ERRADDR7	ECC Error Address 7 Register
L2CTL_ERRADDR8	ECC Error Address 8 Register
L2CTL_ET0	Error Type 0 Register
L2CTL_ET1	Error Type 1 Register
L2CTL_INIT	Initialization Register
L2CTL_ISTAT	Initialization Status Register
L2CTL_PCTL	Power Control Register
L2CTL_REVID	Revision ID Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_SADR	Scrub Start Address Register
L2CTL_SCNT	Scrub Count Register
L2CTL_SCTL	Scrub Control Register
L2CTL_STAT	Status Register
L2CTL_WPCR	Write Priority Count Register

Control Register

The `L2CTL_CTL` register includes a write protection bit, enables L2 banks, and selects mapping of banks (as ECC RAM or data RAM).

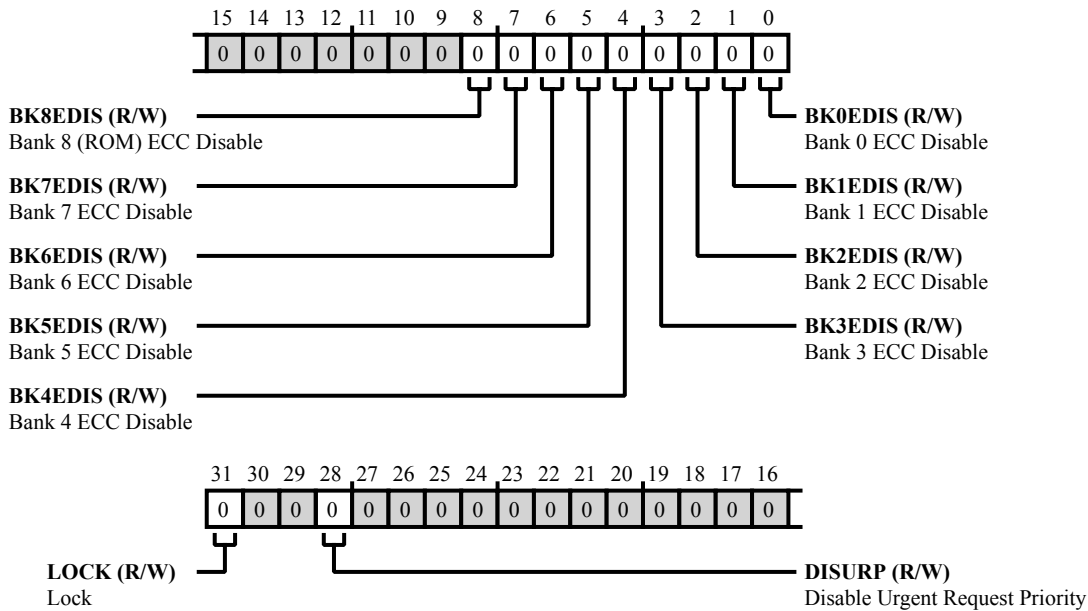


Figure 8-5: L2CTL_CTL Register Diagram

Table 8-7: L2CTL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
		If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>L2CTL_CTL.LOCK</code> bit is set, the <code>L2CTL_CTL</code> register is read only (locked).
		0 Unlock 1 Lock
28 (R/W)	DISURP	Disable Urgent Request Priority.
		The <code>L2CTL_CTL.DISURP</code> disables urgent request priority mode for all L2 banks.
		0 Enable URP 1 Disable URP
8 (R/W)	BK8EDIS	Bank 8 (ROM) ECC Disable.
		The <code>L2CTL_CTL.BK8EDIS</code> bit disables L2 bank 8 (ROM) ECC operation.
		0 Enable ECC 1 Disable ECC

Table 8-7: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	BK7EDIS	Bank 7 ECC Disable. The L2CTL_CTL.BK7EDIS bit disables L2 bank 7 ECC operation.
		0 Enable ECC
		1 Disable ECC
6 (R/W)	BK6EDIS	Bank 6 ECC Disable. The L2CTL_CTL.BK6EDIS bit disables L2 bank 6 ECC operation.
		0 Enable ECC
		1 Disable ECC
5 (R/W)	BK5EDIS	Bank 5 ECC Disable. The L2CTL_CTL.BK5EDIS bit disables L2 bank 5 ECC operation.
		0 Enable ECC
		1 Disable ECC
4 (R/W)	BK4EDIS	Bank 4 ECC Disable. The L2CTL_CTL.BK4EDIS bit disables L2 bank 4 ECC operation.
		0 Enable ECC
		1 Disable ECC
3 (R/W)	BK3EDIS	Bank 3 ECC Disable. The L2CTL_CTL.BK3EDIS bit disables L2 bank 3 ECC operation.
		0 Enable ECC
		1 Disable ECC
2 (R/W)	BK2EDIS	Bank 2 ECC Disable. The L2CTL_CTL.BK2EDIS bit disables L2 bank 2 ECC operation.
		0 Enable ECC
		1 Disable ECC
1 (R/W)	BK1EDIS	Bank 1 ECC Disable. The L2CTL_CTL.BK1EDIS bit disables L2 bank 1 ECC operation.
		0 Enable ECC
		1 Disable ECC

Table 8-7: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	BK0EDIS	Bank 0 ECC Disable. The L2CTL_CTL.BK0EDIS bit disables L2 bank 0 ECC operation.	
		0	Enable ECC
		1	Disable ECC

Error Type 0 Address Register

The `L2CTL_EADDR0` register holds the address that created an access error on the L2 port 0 bus interface. This register is updated only if the corresponding error status bit (`L2CTL_STAT.ERR0`) is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_EADDR0` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

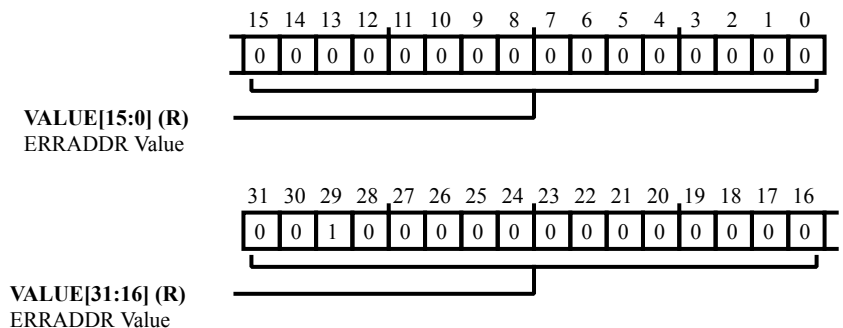


Figure 8-6: L2CTL_EADDR0 Register Diagram

Table 8-8: L2CTL_EADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_EADDR0.VALUE</code> bits hold the address causing the bus error.

Error Type 1 Address Register

The `L2CTL_EADDR1` register holds the address that created an access error on the L2 port 1 bus interface. This register is updated only if the corresponding error status bit (`L2CTL_STAT.ERR1`) is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_EADDR1` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

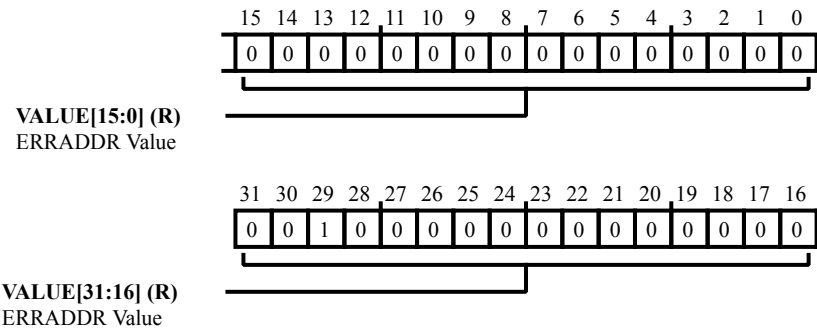


Figure 8-7: L2CTL_EADDR1 Register Diagram

Table 8-9: L2CTL_EADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_EADDR1.VALUE</code> bits hold the address causing the bus error.

ECC Error Address 0 Register

The `L2CTL_ERRADDR0` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank’s status bit (`L2CTL_STAT.ECCERR0`) is cleared. After the bank’s status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

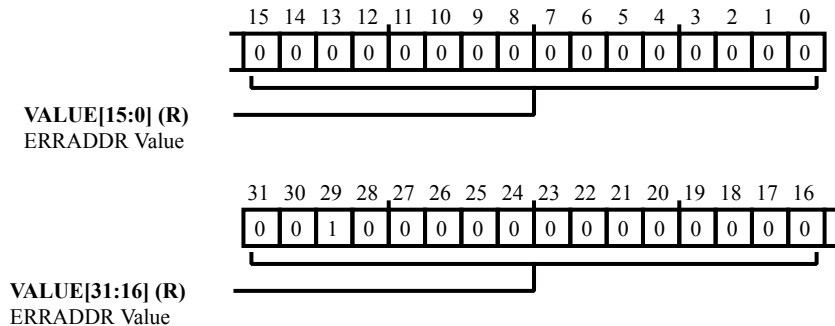


Figure 8-8: L2CTL_ERRADDR0 Register Diagram

Table 8-10: L2CTL_ERRADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR0.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 1 Register

The `L2CTL_ERRADDR1` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR1`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

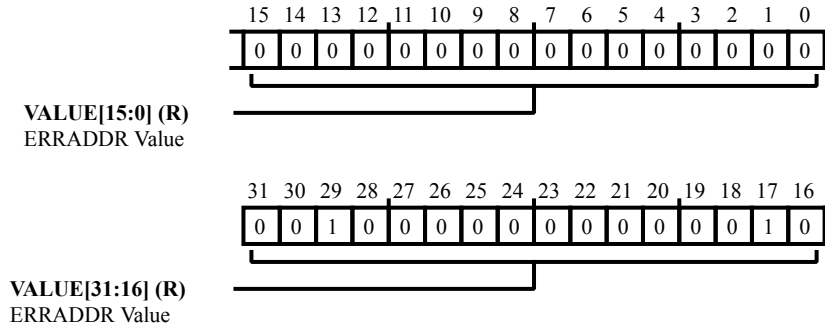


Figure 8-9: L2CTL_ERRADDR1 Register Diagram

Table 8-11: L2CTL_ERRADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR1.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 2 Register

The `L2CTL_ERRADDR2` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR2`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

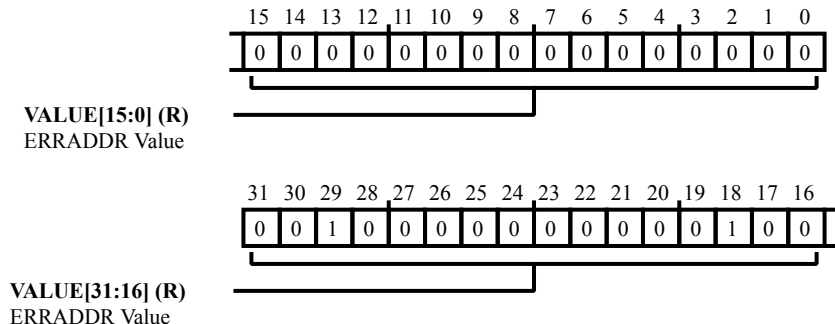


Figure 8-10: L2CTL_ERRADDR2 Register Diagram

Table 8-12: L2CTL_ERRADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR2.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 3 Register

The `L2CTL_ERRADDR3` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR3`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

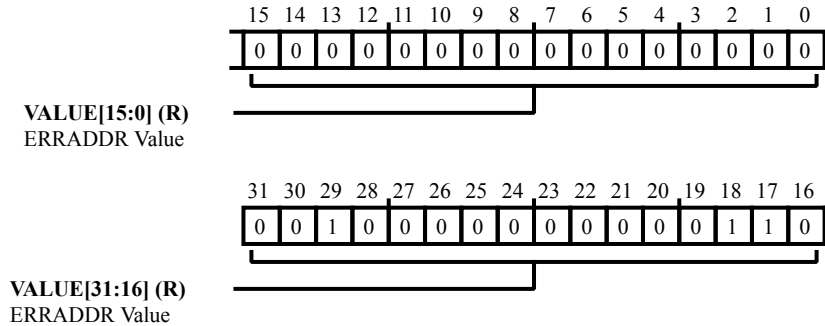


Figure 8-11: L2CTL_ERRADDR3 Register Diagram

Table 8-13: L2CTL_ERRADDR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR3.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 4 Register

The `L2CTL_ERRADDR4` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank’s status bit (`L2CTL_STAT.ECCERR4`) is cleared. After the bank’s status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

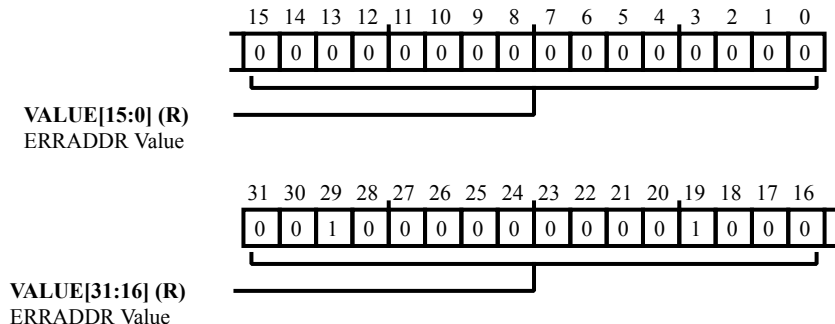


Figure 8-12: L2CTL_ERRADDR4 Register Diagram

Table 8-14: L2CTL_ERRADDR4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR4.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 5 Register

The `L2CTL_ERRADDR5` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank’s status bit (`L2CTL_STAT.ECCERR5`) is cleared. After the bank’s status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

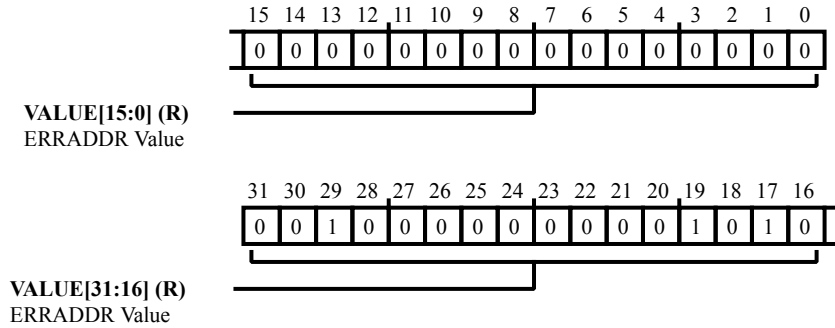


Figure 8-13: L2CTL_ERRADDR5 Register Diagram

Table 8-15: L2CTL_ERRADDR5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR5.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 6 Register

The `L2CTL_ERRADDR6` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR6`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

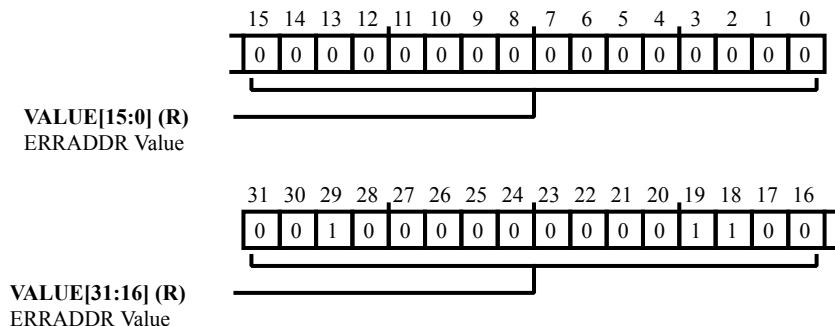


Figure 8-14: L2CTL_ERRADDR6 Register Diagram

Table 8-16: L2CTL_ERRADDR6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR6.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 7 Register

The `L2CTL_ERRADDR7` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR7`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

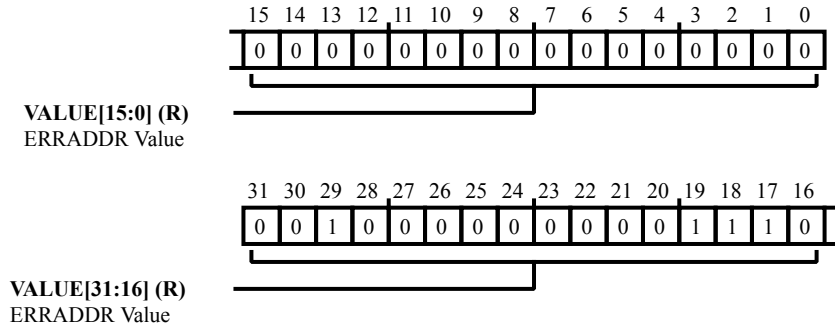


Figure 8-15: L2CTL_ERRADDR7 Register Diagram

Table 8-17: L2CTL_ERRADDR7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR7.VALUE</code> bits hold the address containing the ECC double-bit error.

ECC Error Address 8 Register

The `L2CTL_ERRADDR8` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank’s status bit (`L2CTL_STAT.ECCERR8`) is cleared. After the bank’s status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

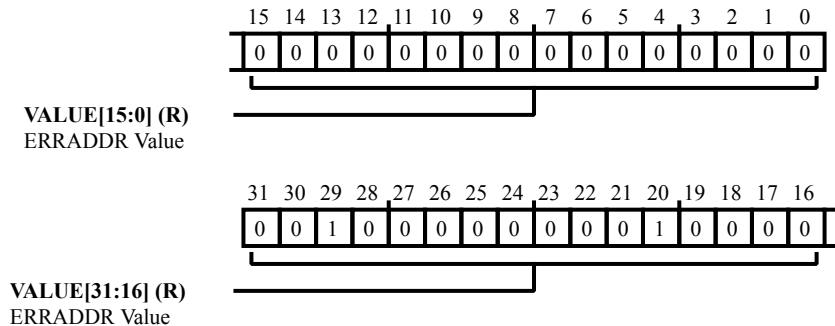


Figure 8-16: L2CTL_ERRADDR8 Register Diagram

Table 8-18: L2CTL_ERRADDR8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The <code>L2CTL_ERRADDR8.VALUE</code> bits hold the address containing the ECC double-bit error.

Error Type 0 Register

The `L2CTL_ET0` register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port 0. This register is updated only if the corresponding error status bit `L2CTL_STAT.ERR0` is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_ET0` register until a `W1C` clears the corresponding status bit. If read and write access errors occur simultaneously, the `L2CTL_ET0` captures the write access error, keeping in sync with the error address register (`L2CTL_EADDR0`).

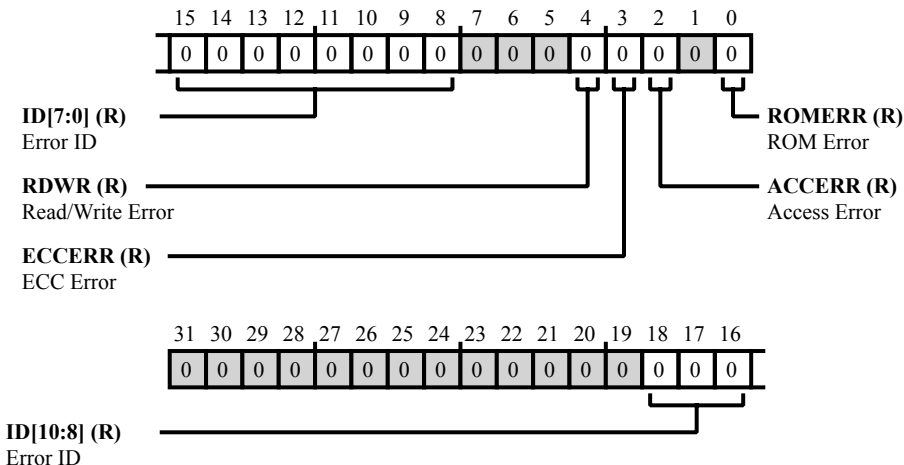


Figure 8-17: L2CTL_ET0 Register Diagram

Table 8-19: L2CTL_ET0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:8 (R/NW)	ID	Error ID. The <code>L2CTL_ET0.ID</code> bits hold the bus master ID of the access that caused an error.
4 (R/NW)	RDWR	Read/Write Error. The <code>L2CTL_ET0.RDWR</code> bit indicates whether a read or write access caused an error.
		0 Read Access created Error
		1 Write Access created Error
3 (R/NW)	ECCERR	ECC Error. The <code>L2CTL_ET0.ECCERR</code> bit indicates whether the access had an ECC double-bit error.
2 (R/NW)	ACCERR	Access Error. The <code>L2CTL_ET0.ACCERR</code> bit indicates whether the access went to a restricted bank.
0 (R/NW)	ROMERR	ROM Error. The <code>L2CTL_ET0.ROMERR</code> bit indicates whether a write access went to a ROM area.

Error Type 1 Register

The `L2CTL_ET1` register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port 1. This register is updated only if the corresponding error status bit `L2CTL_STAT.ERR1` is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_ET1` register until a `W1C` clears the corresponding status bit. If read and write access errors occur simultaneously, the `L2CTL_ET1` captures the write access error, keeping in sync with the error address register (`L2CTL_EADDR1`).

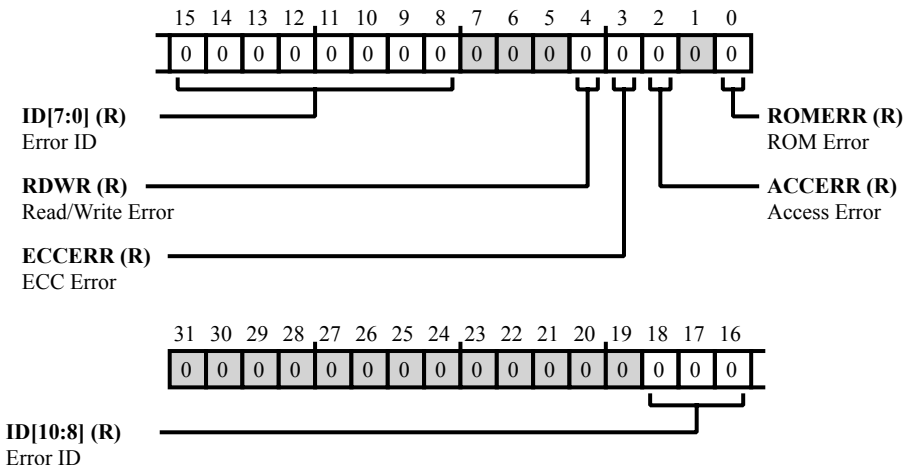


Figure 8-18: L2CTL_ET1 Register Diagram

Table 8-20: L2CTL_ET1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:8 (R/NW)	ID	Error ID. The <code>L2CTL_ET1.ID</code> bits hold the bus master ID of the access that caused an error.
4 (R/NW)	RDWR	Read/Write Error. The <code>L2CTL_ET1.RDWR</code> bit indicates whether a read or write access caused an error.
		0 Read access created error
		1 Write access created error
3 (R/NW)	ECCERR	ECC Error. If the <code>L2CTL_ET1.ECCERR</code> bit =1, the access had an ECC double-bit error.
2 (R/NW)	ACCERR	Access Error. If the <code>L2CTL_ET1.ACCERR</code> bit =1, the access went to a restricted bank.
0 (R/NW)	ROMERR	ROM Error. If the <code>L2CTL_ET1.ROMERR</code> bit =1, a write access went to a ROM area.

Initialization Register

The `L2CTL_INIT` register initializes memory banks with 64'b0 and ECC bits corresponding to 64'b0. Any writes to the bits in this register while initialization is occurring to any of the banks is ignored. All bits are W1A (Write 1 for Action).

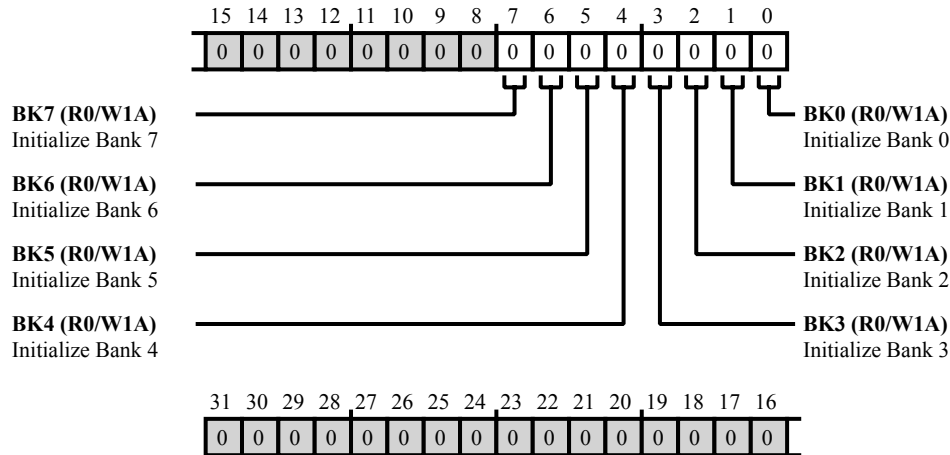


Figure 8-19: L2CTL_INIT Register Diagram

Table 8-21: L2CTL_INIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R0/W1A)	BK7	Initialize Bank 7. The write-1 to the <code>L2CTL_INIT.BK7</code> bit initializes bank 7 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
6 (R0/W1A)	BK6	Initialize Bank 6. The write-1 to the <code>L2CTL_INIT.BK6</code> bit initializes bank 6 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
5 (R0/W1A)	BK5	Initialize Bank 5. The write -1 to the <code>L2CTL_INIT.BK5</code> bit initializes bank 5 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
4 (R0/W1A)	BK4	Initialize Bank 4. The write-1 to the <code>L2CTL_INIT.BK4</code> bit initializes bank 4 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.

Table 8-21: L2CTL_INIT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R0/W1A)	BK3	Initialize Bank 3. The write-1 to the L2CTL_INIT.BK3 bit initializes bank 3 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
2 (R0/W1A)	BK2	Initialize Bank 2. The write-1 to the L2CTL_INIT.BK2 bit initializes bank 2 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
1 (R0/W1A)	BK1	Initialize Bank 1. The write-1 to the L2CTL_INIT.BK1 bit initializes bank 1 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.
0 (R0/W1A)	BK0	Initialize Bank 0. The write-1 to the L2CTL_INIT.BK0 bit initializes bank 0 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit during the initialization of the banks is ignored.

Initialization Status Register

The `L2CTL_ISTAT` register holds the status of the RAM bank initialization. If set, the corresponding bank is initialized.

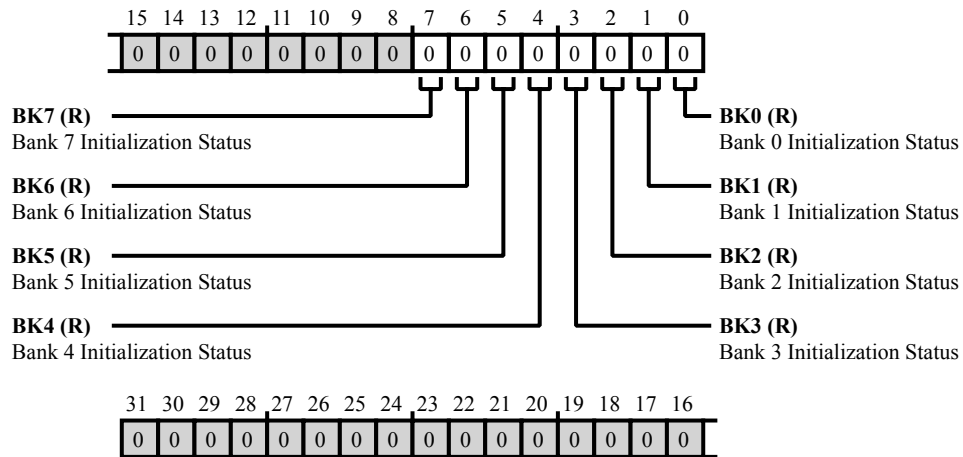


Figure 8-20: L2CTL_ISTAT Register Diagram

Table 8-22: L2CTL_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	BK7	Bank 7 Initialization Status. The <code>L2CTL_ISTAT.BK7</code> bits hold the bank 7 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.
6 (R/NW)	BK6	Bank 6 Initialization Status. The <code>L2CTL_ISTAT.BK6</code> bits hold the bank 6 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.
5 (R/NW)	BK5	Bank 5 Initialization Status. The <code>L2CTL_ISTAT.BK5</code> bits hold the bank 5 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.
4 (R/NW)	BK4	Bank 4 Initialization Status. The <code>L2CTL_ISTAT.BK4</code> bits hold the bank 4 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.
3 (R/NW)	BK3	Bank 3 Initialization Status. The <code>L2CTL_ISTAT.BK3</code> bits hold the bank 3 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.
2 (R/NW)	BK2	Bank 2 Initialization Status. The <code>L2CTL_ISTAT.BK2</code> bits hold the bank 2 initialization status. A W1A on a <code>BKxINIT</code> bit clears this bit and the bit is set when initialization completes.

Table 8-22: L2CTL_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	BK1	Bank 1 Initialization Status. The L2CTL_ISTAT.BK1 bits hold the bank 1 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
0 (R/NW)	BK0	Bank 0 Initialization Status. The L2CTL_ISTAT.BK0 bits hold the bank 0 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.

Power Control Register

The `L2CTL_PCTL` register has the various control settings for selectively enabling the deep sleep and shut down power saving features.

NOTE: The corresponding L2 bank should not be accessed if the power control feature is enabled for that bank. An access to a bank in deep sleep/shut down may create unpredictable behavior.

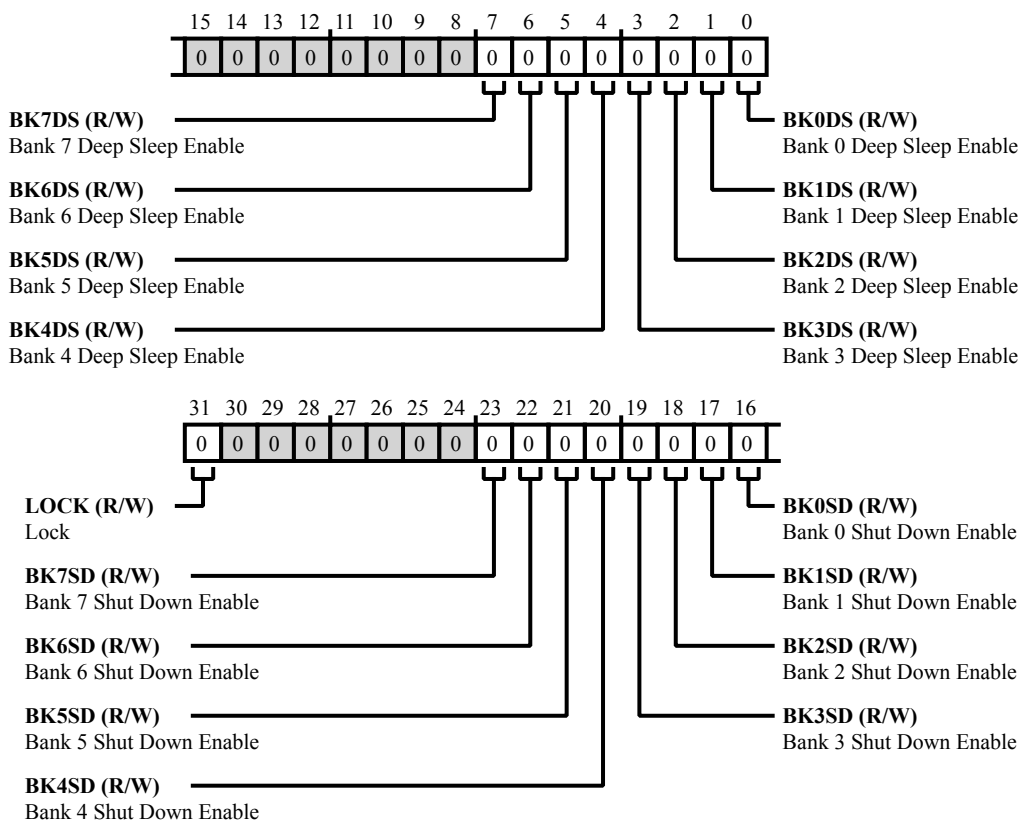


Figure 8-21: L2CTL_PCTL Register Diagram

Table 8-23: L2CTL_PCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>L2CTL_PCTL.LOCK</code> bit is set, the <code>L2CTL_PCTL</code> register is read only (locked).
		0 Unlock
		1 Lock
23 (R/W)	BK7SD	Bank 7 Shut Down Enable. The <code>L2CTL_PCTL.BK7SD</code> bits enables bank 7 shut down.

Table 8-23: L2CTL_PCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	BK6SD	Bank 6 Shut Down Enable. The L2CTL_PCTL.BK6SD bits enables bank 6 shut down.
21 (R/W)	BK5SD	Bank 5 Shut Down Enable. The L2CTL_PCTL.BK5SD bits enables bank 5 shut down.
20 (R/W)	BK4SD	Bank 4 Shut Down Enable. The L2CTL_PCTL.BK4SD bits enables bank 4 shut down.
19 (R/W)	BK3SD	Bank 3 Shut Down Enable. The L2CTL_PCTL.BK3SD bits enables bank 3 shut down.
18 (R/W)	BK2SD	Bank 2 Shut Down Enable. The L2CTL_PCTL.BK2SD bits enables bank 2 shut down.
17 (R/W)	BK1SD	Bank 1 Shut Down Enable. The L2CTL_PCTL.BK1SD bits enables bank 1 shut down.
16 (R/W)	BK0SD	Bank 0 Shut Down Enable. The L2CTL_PCTL.BK0SD bits enables bank 0 shut down.
7 (R/W)	BK7DS	Bank 7 Deep Sleep Enable. The L2CTL_PCTL.BK7DS bits enables bank 7 deep sleep.
6 (R/W)	BK6DS	Bank 6 Deep Sleep Enable. The L2CTL_PCTL.BK6DS bits enables bank 6 deep sleep.
5 (R/W)	BK5DS	Bank 5 Deep Sleep Enable. The L2CTL_PCTL.BK5DS bits enables bank 5 deep sleep.
4 (R/W)	BK4DS	Bank 4 Deep Sleep Enable. The L2CTL_PCTL.BK4DS bits enables bank 4 deep sleep.
3 (R/W)	BK3DS	Bank 3 Deep Sleep Enable. The L2CTL_PCTL.BK3DS bits enables bank 3 deep sleep.
2 (R/W)	BK2DS	Bank 2 Deep Sleep Enable. The L2CTL_PCTL.BK2DS bits enables bank 2 deep sleep.
1 (R/W)	BK1DS	Bank 1 Deep Sleep Enable. The L2CTL_PCTL.BK1DS bits enables bank 1 deep sleep.
0 (R/W)	BK0DS	Bank 0 Deep Sleep Enable. The L2CTL_PCTL.BK0DS bits enables bank 0 deep sleep.

Revision ID Register

The `L2CTL_REVID` register provides the L2 Revision ID.

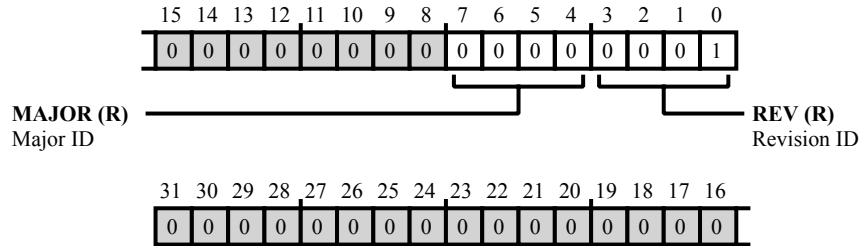


Figure 8-22: L2CTL_REVID Register Diagram

Table 8-24: L2CTL_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major ID. The <code>L2CTL_REVID.MAJOR</code> bit indicates L2 Major ID.
3:0 (R/NW)	REV	Revision ID. The <code>L2CTL_REVID.REV</code> bit indicates the L2 Revision ID.

Read Priority Count Register

The `L2CTL_RPCR` register stores the count value to be used for priority elevation for bus read channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SCLK cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus read channel. A best practice is to program this register before initiating an L2 access.

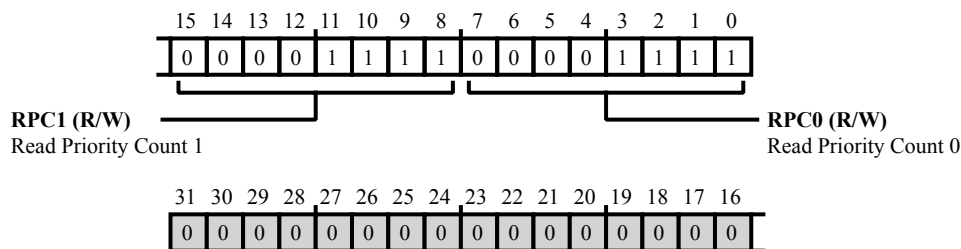


Figure 8-23: L2CTL_RPCR Register Diagram

Table 8-25: L2CTL_RPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	RPC1	Read Priority Count 1. The <code>L2CTL_RPCR.RPC1</code> bits hold the priority count for L2 bus read channel 1.
7:0 (R/W)	RPC0	Read Priority Count 0. The <code>L2CTL_RPCR.RPC0</code> bits hold the priority count for L2 bus read channel 0.

Scrub Start Address Register

The `L2CTL_SADR` register stores the scrub start address value. Writes to this register can be prevented by setting the `L2CTL_SCTL.LOCK` bit and the global lock.

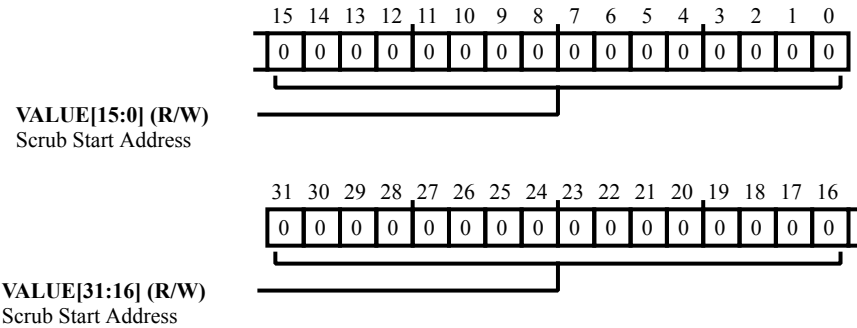


Figure 8-24: L2CTL_SADR Register Diagram

Table 8-26: L2CTL_SADR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Scrub Start Address. The <code>L2CTL_SADR.VALUE</code> bits hold the scrub start address. The writes to this register can be prevented by setting the <code>L2_SCTL.LOCK</code> and the global lock.

Scrub Count Register

The `L2CTL_SCNT` register determines the number of 64-bit locations scrubbed starting from the start address (`L2CTL_SADR` register). Writes to the `L2CTL_SCNT` register can be prevented by setting the `L2CTL_CTL.LOCK` bit and the global lock.

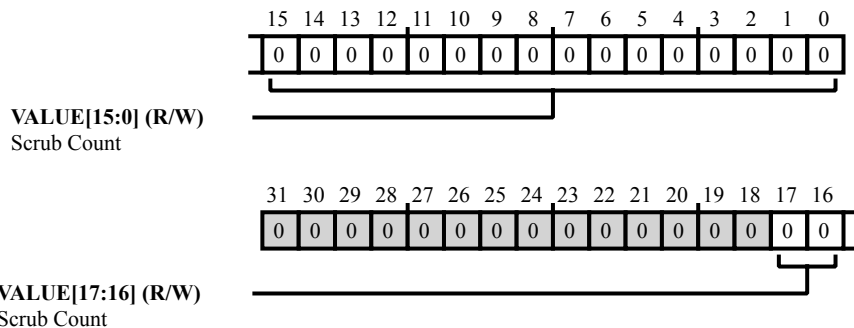


Figure 8-25: L2CTL_SCNT Register Diagram

Table 8-27: L2CTL_SCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:0 (R/W)	VALUE	Scrub Count. The <code>L2CTL_SCNT.VALUE</code> bits determines the number of 64-bit locations scrubbed starting from the start address. Ensure value programmed is less than the total addressable 64-bit L2 RAM locations available.

Scrub Control Register

The `L2CTL_SCTL` register holds the automatic scrub related controls. This is a read/write register. Memory scrub can be performed by programming the `L2CTL_SADR` (scrub start address) register with the start address of the scrub and the `L2CTL_SCNT` register with the total number of 64-bit addresses to be scrubbed starting from the `L2CTL_SADR` register. The number of cycles between each scrub can be programmed with the `L2CTL_SCTL.SRT` bit.

During the scrub the controller issues a read followed by write back if there is a single bit ECC error. Once the L2 controller completes scrubbing the memory region mentioned using the `L2CTL_SADR` and the `L2CTL_SCNT` registers, an interrupt is generated and scrubbing re-starts from the start unless the scrub enable bit is disabled in the control register. If scrub enable is cleared before completing the address range used in the `L2CTL_SADR` and `L2CTL_SCNT` registers, the scrub stops after completing any already issued scrub access.

The scrub read/writes always start from the full 64-bit equivalent of the address written into the `L2CTL_SADR` register. A 64-bit value is always read and the 64-bit value is written back. The scrub access has the highest priority. Programs can configure 8-, 16-, or 32-bit addresses in the `L2CTL_SADR` register but the lower 3 bits are treated as don't care because the internal memory array is always accessed in a 64-bit mode.

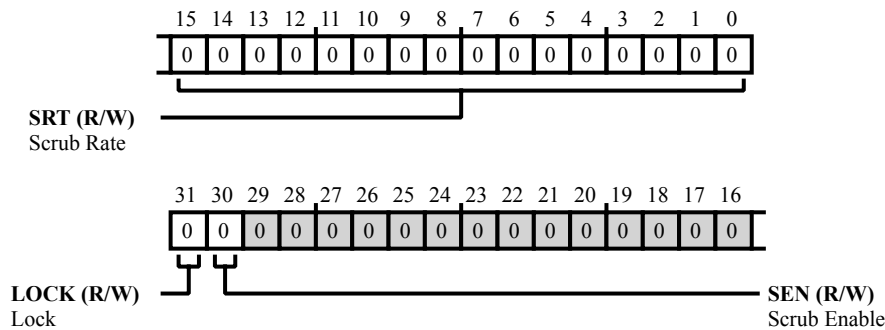


Figure 8-26: L2CTL_SCTL Register Diagram

Table 8-28: L2CTL_SCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>L2CTL_SCTL.LOCK</code> bit is set, the <code>L2CTL_SCTL</code> register is read only (locked).
		0 Unlock
		1 Lock
30 (R/W)	SEN	Scrub Enable. The <code>L2CTL_SCTL.SEN</code> bits enable automatic scrub.

Table 8-28: L2CTL_SCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SRT	Scrub Rate. The L2CTL_SCTL.SRT bits determines the number of clock cycles that elapsed between two automatic scrubs.

Status Register

The `L2CTL_STAT` register indicates ECC error status, refresh register status, and bus error status of Port0 and Port1.

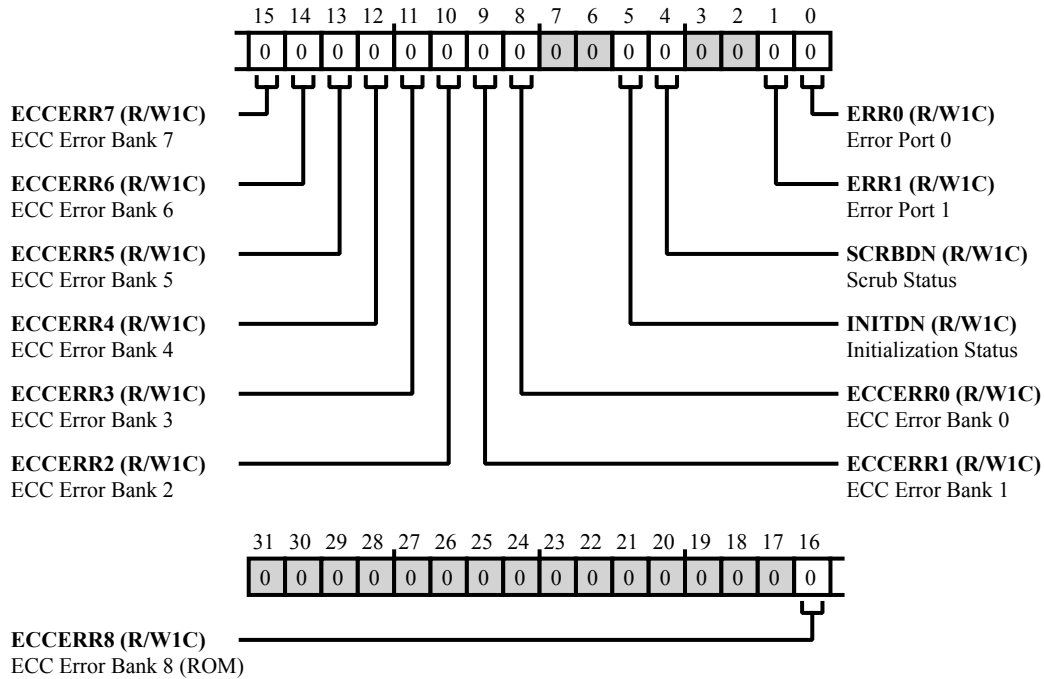


Figure 8-27: L2CTL_STAT Register Diagram

Table 8-29: L2CTL_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W1C)	ECCERR8	ECC Error Bank 8 (ROM). The <code>L2CTL_STAT.ECCERR8</code> bit indicates that an ECC double-bit error occurred inside L2 bank 8 (ROM).
		0 No Status
		1 ECC Double Bit Error
15 (R/W1C)	ECCERR7	ECC Error Bank 7. The <code>L2CTL_STAT.ECCERR7</code> bit indicates that an ECC double-bit error occurred inside L2 bank 7.
		0 No Status
		1 ECC Double Bit Error

Table 8-29: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	ECCERR6	ECC Error Bank 6. The L2CTL_STAT.ECCERR6 bit indicates that an ECC double-bit error occurred inside L2 bank 6.
		0 No Status
		1 ECC Double Bit Error
13 (R/W1C)	ECCERR5	ECC Error Bank 5. The L2CTL_STAT.ECCERR5 bit indicates that an ECC double-bit error occurred inside L2 bank 5.
		0 No Status
		1 ECC Double Bit Error
12 (R/W1C)	ECCERR4	ECC Error Bank 4. The L2CTL_STAT.ECCERR4 bit indicates that an ECC double-bit error occurred inside L2 bank 4.
		0 No Status
		1 ECC Double Bit Error
11 (R/W1C)	ECCERR3	ECC Error Bank 3. The L2CTL_STAT.ECCERR3 bit indicates that an ECC double-bit error occurred inside L2 bank 3.
		0 No Status
		1 ECC Double Bit Error
10 (R/W1C)	ECCERR2	ECC Error Bank 2. The L2CTL_STAT.ECCERR2 bit indicates that an ECC double-bit error occurred inside L2 bank 2.
		0 No Status
		1 ECC Double Bit Error
9 (R/W1C)	ECCERR1	ECC Error Bank 1. The L2CTL_STAT.ECCERR1 bit indicates that an ECC double-bit error occurred inside L2 bank 1.
		0 No Status
		1 ECC Double Bit Error

Table 8-29: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	ECCERR0	ECC Error Bank 0. The L2CTL_STAT.ECCERR0 bit indicates that an ECC double-bit error occurred inside L2 bank 0.
		0 No Status
		1 ECC Double Bit Error
5 (R/W1C)	INITDN	Initialization Status. The L2CTL_STAT.INITDN bit indicates whether initialization has been completed.
		0 Initialization Not Complete
		1 Initialization Completed
4 (R/W1C)	SCRBDN	Scrub Status. The L2CTL_STAT.SCRBDN bit indicates whether a round of memory scrub has completed.
		0 Scrub Not Complete
		1 Scrub Completed
1 (R/W1C)	ERR1	Error Port 1. The L2CTL_STAT.ERR1 indicates whether the L2CTL has detected a bus access error on L2s bus port 1.
		0 No Error
		1 Bus Access Error
0 (R/W1C)	ERR0	Error Port 0. The L2CTL_STAT.ERR0 indicates whether the L2CTL has detected a bus access error on L2s bus port 0.
		0 No Error
		1 Bus Access Error

Write Priority Count Register

The `L2CTL_WPCR` register stores the count value to be used for priority elevation for bus write channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SCLK cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus write channel. A best practice is to program this register before initiating an L2 access.

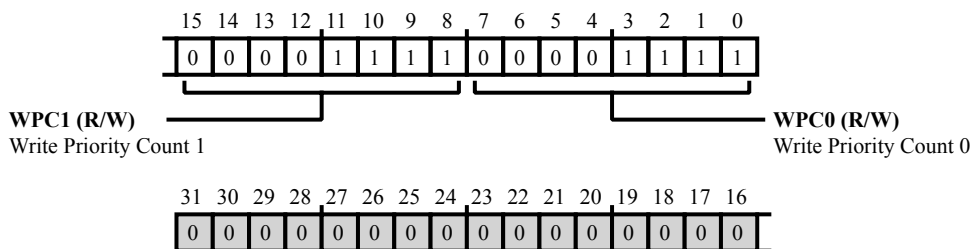


Figure 8-28: L2CTL_WPCR Register Diagram

Table 8-30: L2CTL_WPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	WPC1	Write Priority Count 1. The <code>L2CTL_WPCR.WPC1</code> bits hold the priority count for L2 bus write channel 1.
7:0 (R/W)	WPC0	Write Priority Count 0. The <code>L2CTL_WPCR.WPC0</code> bits hold the priority count for L2 bus write channel 0.

9 Dynamic Memory Controller (DMC)

The dynamic memory controller (DMC) provides a glueless interface between DDR3 SDRAMs and the system crossbar interface (SCB). The DMC enables execution of instructions from, as well as transfer of data to and from, DDR3 SDRAM respectively.

NOTE: The term DDR3 is referred to generically as DDR SDRAM in the rest of this chapter unless otherwise noted.

The DMC is partitioned in a manner that allows reconfiguration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the *protocol controller*. An access and operation reordering mechanism is incorporated as an *efficiency controller*. An SCB slave interface is provided to interface with the on-chip interconnect. This interface results in an efficient slave implementation owing to its out-of-order transaction capabilities. The control and status registers present in the DMC can be accessed using the MMR access bus.

The DMC supports access to the external memory by core and DMA accesses.

DMC Features

The DMC includes a protocol controller that supports:

- JESD79-3E compatible double data rate DDR3 SDRAM devices

The features of the dynamic memory controller are:

- Provides 16-bit data only interface to SDRAM devices
- Supports a single external rank (one chip select)
- Provides page hit detection that supports multiple column accesses to the same row
- User-specified active, precharge, and refresh commands.
- Programmable SDRAM access timing parameters
- Enables automatic refresh generation with programmable refresh intervals
- Self-refresh mode to reduce system power consumption
- Efficient transaction processing to improve throughput and bandwidth using:

- Software programmable SCB IDs to allow SCB ID-based priority
- The ability to postpone up to eight auto-refresh commands
- Software selectable closed page scheme on a per bank basis
- Simple transaction scheduling mechanism to reduce read write turnaround frequency on the memory bus
- Accesses with the same SCB ID are scheduled back-to-back to take advantage of same page access in SDRAM
- Caching of SDRAM read data burst for specific masters to reduce the latency for same burst accesses.

The DDR3 features are:

- 512 Mb to 8 Gb device sizes
- Burst length BL = 8
- Support for additive latency
- Support for programmable (and ZQ calibration) ODT and drive impedance
- Support for read leveling
- Support for write leveling

Feature Exclusions

The DMC exclusions are as follows:

For DDR3:

- 4-bit and 8-bit wide DDR3 DRAM memories are not supported
- Burst interleaved accesses are not supported
- Both burst chop-BC4 and BC4-on-the-fly are not supported
- Auto refresh pull-in is not supported
- DLL off mode is not supported

DMC Functional Description

The dynamic memory controller consists of master and slave interfaces, a protocol controller, and an efficiency controller. The following sections describe the function of these interfaces and controllers.

ADSP-2156x DMC Register List

The Dynamic Memory Controller module (DMC) provides an interface to external double-data-rate SDRAM. This interface supports various DDR standards (see chapter descriptions). A set of registers governs DMC controller operations. For more information on DMC controller functionality, see the DMC Controller Register Descriptions.

Table 9-1: ADSP-2156x DMC Register List

Name	Description
DMC_CFG	Configuration Register
DMC_CTL	Control Register
DMC_DLLCTL	DLL Control Register
DMC_DT_CALIB_ADDR	Data Calibration Address Register
DMC_DT_DATA_CALIB_DATA0	Data Calibration Data 0 Register
DMC_DT_DATA_CALIB_DATA1	Data Calibration Data 1 Register
DMC_EFFCTL	Efficiency Control Register
DMC_EMR3	Shadow EMR3 Register
DMC_MR	Shadow MR0 Register (DDR3)
DMC_MR1	Shadow MR1 Register (DDR3)
DMC_MR2	Shadow MR2 Register (DDR3)
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

ADSP-2156x DMC Register List

DMCPHY (DMC) contains the following registers.

Table 9-2: ADSP-2156x DMC Register List

Name	Description
DMC_DDR_CA_CTL	DDR CA Lane Control Register
DMC_DDR_LANE0_CTL0	Data Lane 0 Control Register 0
DMC_DDR_LANE0_CTL1	Data Lane 0 Control Register 1
DMC_DDR_LANE1_CTL0	Data Lane 1 Control Register 0
DMC_DDR_LANE1_CTL1	Data Lane 1 Control Register 1
DMC_DDR_ROOT_CTL	DDR ROOT Module Control Register
DMC_DDR_ZQ_CTL0	DDR Calibration Control Register 0
DMC_DDR_ZQ_CTL1	DDR Calibration Control Register 1
DMC_DDR_ZQ_CTL2	DDR Calibration Control Register 2

Protocol Controller

The DDR SDRAM protocol controller translates memory access requests from the SCB (system crossbar) interface to JEDEC protocol-specific transactions used by DDR SDRAM devices.

The protocol controller ensures that the various timing parameters are met before reading and writing the SDRAM. The controller also performs the SDRAM initialization sequence as mandated by the standard. The protocol controller can:

- Issue reads and writes
- Precharge a row in a bank
- Activate a row in a bank
- Put the SDRAM devices in self-refresh and power-down modes

The protocol controller takes mode register writes from the MMR interface and translates them into mode register writes to SDRAM. Writing into the mode register is restricted through a mask register.

Efficiency Controller

The efficiency controller controls the ordering of transfers buffered in the read and write command buffers. It attempts to order transfers to optimize the available memory bandwidth. The DMC uses a number of schemes, described in the following sections, to increase the throughput.

Page-Based Scheduling

The DMC parses each write and read transaction that it buffered and gets the information of the row (page) and bank address. The protocol controller maintains the information about the pages that are opened in each bank. The efficiency controller uses the information about the opened pages while scheduling the buffered transactions. The transactions to the opened pages are given higher priority than the other outstanding transactions.

Same Master Transaction Scheduling

The DMC also stores the ID of each transaction that it buffered. In most of the cases, the transactions related to a master result in page hits from the locality of reference rule. The efficiency controller uses the ID information of the transactions while scheduling. When the page-based scheduling of the buffered transactions is complete, same master transaction scheduling is triggered. If multiple transactions from a master are received, the efficiency controller schedules the transactions back-to-back.

DMC Read Data Buffer

The DMC read data buffer contains a data buffer and an address buffer. The depth of the data buffer is equal to the burst length that is programmed in SDRAM. The address buffer holds the corresponding SDRAM burst address. When an SDRAM write address from any master matches an address in the DMC read data buffer, the DMC invalidates the related data in the read buffer. When the `DMC_RDDATABUFMSK1` or `DMC_RDDATABUFMSK2` register is programmed with a value other than zero, the DMC read data buffer operation is enabled. The set of masters whose data is buffered and retrieved are programmed in the `DMC_RDDATABUFID1` or `DMC_RDDATABUFID2` registers. The DMC can use the `DMC_RDDATABUFMSK1` and `DMC_RDDATABUFMSK2` ID registers to select a set of masters similar to the programming of the `DMC_PRIOMSK` and `DMC_PRIOMSK2` registers.

See the [SCB ID-Based Priority](#) section for details.

Closed Page Per Bank

The `DMC_EFFCTL` register provides per-bank granularity for closing pages. The software can determine that most accesses to a given bank in memory always result in a missed page. In this case, set the `PREC_BANK` bit corresponding to the required bank to close the row after every transfer. This proactive step can result in reduced thrashing and increases memory throughput.

SCB ID-Based Priority

The primary goal of the dynamic memory controller is to improve sustainable memory system bandwidth so that the service time for the average request can be reduced. However, to service critical requests from any master in the system, the DMC provides a mechanism to elevate priority of a given access. The DMC priority ID registers (`DMC_PRIO` and `DMC_PRIO2`) can be programmed with up to two SCB IDs with elevated priority.

After every access in a snapshot, the command buffers are searched to determine whether an ID of a command matches with the ID programmed in the `DMC_PRIO` and `DMC_PRIO2` registers. The priority SCB ID access is sent before the subsequent access in the snapshot if:

- A match occurs, and
- The direction of the access (for example write) is the same as the direction of the snapshot (write)

There is an alternative to providing priority to a specific SCB ID. If a number of IDs from the same master require priority, program the DMC priority mask ID registers (`DMC_PRIOMSK` and `DMC_PRIOMSK2`) so that the corresponding bits are 0. The DMC uses a combination of the `DMC_PRIO` and `DMC_PRIO2` registers and the `DMC_PRIOMSK/DMC_PRIOMSK2` registers to elevate the priority of a select few or all IDs that belong to a master. By default, none of the IDs are prioritized. The following are a few possibilities:

- The `DMC_PRIOMSK` field is set to `0x00000000`. If a single ID (7234) needs priority, set the `DMC_PRIOMSK` field to `0xFFFFFFFF` and set the `DMC_PPIO` field to 7234.
- If the `DMC_PRIOMSK` field is set to `0xFFFFFFFFE`, the SCB IDs 7234 and 7235 are given priority.
- If the `DMC_PRIOMSK` field is set to `0xFFFFFFFFC`, the SCB IDs 7234, 7235, 7236, and 7237 are given priority.
- If two transactions with priority, one read and the other a write, are outstanding, the priority transaction that does not change the direction of the DMC access gets priority. The other priority transaction is handled at the beginning of the next snapshot. For example, if a write snapshot is in-progress, the write priority transaction is sent. The read priority transaction is sent at the beginning of the next read snapshot.

NOTE: Use SCB ID-based priority judiciously because it can significantly reduce the throughput of the DMC.

Delaying up to Eight Auto-Refresh Commands

The DMC uses this method to ensure that auto-refresh does not interfere with any critical data transfers. Up to eight auto-refresh commands can accumulate in the DMC. The exact number of auto-refresh commands can be programmed using the `DMC_EFFCTL.NUMREF` bit.

After the first refresh command is accumulated, the DMC constantly looks for an opportunity to schedule a refresh command. When the SCB read and write command buffers become empty for the programmed number of clock cycles (`DMC_EFFCTL.IDLECYC` bit field), the accumulated number of refresh commands are sent back-to-back to the DRAM. (The empty state of the SCB command buffers implies that no access is outstanding.)

After every refresh, the SCB command buffers are checked to ensure that they stay empty. If the SCB command buffers are always full, once the programmed number of refresh commands accumulates, the refresh operation is elevated to urgent priority. One refresh command is sent immediately. After this process, the DMC continues to wait for an opportunity to send out refresh commands. If self-refresh mode is enabled, all pending refresh commands are given out only after that DMC enters into self-refresh mode.

Page and Bank Interleaving

Page and bank interleaving allow consecutive row accesses to fall into the same bank (bank interleaving) or into a different bank (page interleaving). The DMC uses bank interleaving by default (`DMC_CTL.ADDRMODE` bit =0). If the `DMC_CTL.ADDRMODE` bit =1, the DMC uses page interleaving. Page misses in one addressing mode result in hits in the other addressing mode.

System Crossbar Slave Interface

The DMC uses the system crossbar slave interface to move all data. The system crossbar interface accepts interleaved write transactions and sends out-of-order responses. The read and write interfaces consist of buffers for address, data, and control information transferred to or from the system crossbar bus.

The system crossbar interface transactions are sent to the SDRAM only after the SDRAM has been initialized. However, if transactions arrive before or during initialization, they accumulate in the system crossbar interface and are sent out to the protocol controller once the initialization completes.

To increase throughput, the system crossbar write-response is sent out as soon as the final DDR burst is scheduled for transfer into the SDRAM. However, if an auto-refresh is needed, the scheduled write data is sent only after the auto-refresh. A delay can occur. The delay is a maximum of 64 clock cycles from the moment the write response is sent on the SCB to the write operation of the data into SDRAM.

The system crossbar interface performs the following operations:

- Buffers read and write command requests from the system crossbar bus
- Processes the requests by converting them to protocol controller user-interface transfers
- Sends and receives data to or from the protocol controller
- Creates a suitable read/write response and sends read data back to the system crossbar bus

The system crossbar slave interface supports the following:

- All burst lengths (1–16)
- Incremental and wrap bursts
- Data transfer sizes of 8-bit, 16-bit, or 32-bit
- Arrival of write data before write address
- Generation of error responses which include:
 - Any access to an unimplemented region of the external memory space
 - Any access when the SDRAM is in self-refresh mode/power-down mode
 - Any access when the direct command interface is in operation

Read/Write Command and Data Buffers

The system crossbar interface consists of a four-deep read command buffer and a four-deep write command buffer. Up to four write commands and four read commands can be waiting for access to the SDRAM. The system crossbar write buffer is 32 deep. It can support write data interleaving of two. The system crossbar read buffer is 32 deep.

Peripheral Bus Slave Interface

The peripheral bus slave interface connects the dynamic memory controller to the peripheral bus and provides a host controller with access to the registers. The peripheral bus slave interface supports the following features:

- Read and write word accesses
- 32-bit data bus

Architectural Concepts

The following sections provide information on the architecture of the interface.

Controller On Die Termination (ODT)

The controller ODT is enabled with the granularity of a byte lane. The description of this feature can be obtained in the description of the corresponding PHY registers. Controller ODT involves extra overhead in terms of power consumption during reads.

The DMC implements dynamic on die termination at processor pads. When controller ODT is enabled, the termination resistors in the pads are turned on when the controller reads data from the DRAM. These resistors are turned off when the controller writes to the DRAM.

Mode Register Set and Extended Mode Register Set Command

The load mode register command initializes the SDRAM operation parameters. The DMC supports the mode register set and extended mode register set commands. The controller automatically issues the mode register set command during power-on initialization and also when the `DMC_MR` register is written with the `DMC_MSK.MR` bit. The mode register set command is sent after the ongoing data transfer completes.

The DMC automatically issues the mode register set command when the shadow EMR3 register is written. The corresponding `DMC_MSK.EMR3` bits must be enabled.

DDR3 Reset Functionality

DDR3 contains an additional pin corresponding to reset functionality. Reset is part of the initialization sequence but it can be performed asynchronously when needed. The reset procedure is similar to the steps involved in the initialization except the initial part of power-up.

To perform reset on the DDR3 module:

1. Check to ensure the module is in the idle state by polling the `DMC_STAT.IDLE` bit (0x0008).
2. Set the `DMC_CTL.RESET` bit (0x0004).
3. Monitor the `DMC_STAT.RESETDONE` bit for the completion of the reset function.

Do not perform any transactions during a module reset. Wait for the `DMC_STAT.RESETDONE` signal.

DDR3 SDRAM Organization

The DMC supports DDR3 SDRAM memory modules ranging from 512 Mb to 8 Gb. The following tables list the address translation mechanism from the user interface to DDR3 memory interface. The controller also supports two types of addressing modes: bank interleaving (`DMC_CTL.ADDRMODE = 1`) and page interleaving (`DMC_CTL.ADDRMODE = 0`).

Bank Interleaving

The *DDR3 Bank Interleaving* table shows DDR3 bank interleaving.

Table 9-3: DDR3 Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
512 Mb	25:24	23:11	10:1

Table 9-3: DDR3 Bank Interleaving (Continued)

SDRAM size	Bank address bits	Row address bits	Column address bits
1 Gb	26:24	23:11	10:1
2 Gb	27:25	24:11	10:1
8 Gb	29:27	26:11	10:1

Page Interleaving

The *DDR3 Page Interleaving* table shows DDR3 page interleaving.

Table 9-4: DDR3 Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1
2 Gb	27:14	13:11	10:1
4 Gb	28:14	13:11	10.1
8 Gb	29:14	13:11	10.1

DMC Clocking

The DMC uses a divided-down version of the *PLLCLK* (PLL clock) to generate an internal clock for clocking the DMC block and interface. The specific value of the *DCLK* frequency is programmed in the *CGU_DIV* register.

For information on the maximum clock frequency supported for specific modes, refer to the processor data sheet.

NOTE: For details on DMC clocking, see the [Clock Generation Unit \(CGU\)](#) and [Clock Generation Unit \(CGU\)](#) chapters.

DMC DMA

The DMC supports DMA-based transfers to and from external DDR SDRAM memory and internal memory.

The DMC DMA controller, part of the distributed DMA engines (DDE) that are dispersed through the infrastructure, connects to the system crossbar fabric.

The DMC uses two DDEs for memory-to-memory DMA (MDMA). One channel is the source channel, and the second, the destination channel.

DMA transfers on the processor are descriptor-based or register-based. Register-based DMA allows the processor to program DMA control registers directly to initiate a DMA transfer. On completion, the control registers can be automatically updated with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. This transfer allows the chaining together of multiple DMA sequences. In descriptor-based DMA operations, a DMA channel can be programmed to set up and start another DMA transfer automatically after the current sequence completes.

Enhanced DMA operations (such as delay line DMA, scatter or gather DMA) are also supported to or from the DMC module.

DMC Operating Modes

DDR3 Mode

The DMC module supports JESD79-3E compatible double data rate DDR3 SDRAM. To configure this mode of operation, first set (=1) the `DMC_CTL.DDR3EN` bit

Self-Refresh Mode

For low-power consumption, the SDRAM can be put in self-refresh mode. When no data activity occurs, the DMC can put the SDRAM in self refresh to save power. The `DMC_STAT.IDLE` bit indicates the activity on the DMC. If this bit is set, there is no activity in the DMC.

Enable self-refresh mode by writing the `DMC_CTL.SRREQ` bit. The DMC stays in a self-refresh state as long as this bit is asserted. The `DMC_STAT.SRACK` bit indicates when the SDRAM enters self-refresh mode.

When the DMC is in self-refresh mode, the DMC generates an SCB error when any data accesses (read or write requests) is requested.

The DMC can be brought out of self-refresh mode by clearing the `DMC_CTL.SRREQ` bit again. The controller clears the `DMC_STAT.SRACK` bit after the self-refresh operation completes.

DMC Event Control

The DMC has no related interrupt or trigger event information.

DMC Programming Model

The dynamic memory controller contains five groups of memory-mapped registers. The DMC uses the MMR access bus to connect to these registers.

- Control and status registers. These registers control the various operation modes of the dynamic memory controller and provide status.
- Timing parameter registers. The value programmed in these registers depends on the speed grade of the SDRAM device used.
- Mode register mirror registers. These shadow registers are copies of the mode registers residing in the SDRAM device.
- PHY control and status registers. The DMC uses these registers to control the operation of the PHY.
- PAD control registers. The DMC uses these registers to control the various aspects of the I/O pads.

The DMC control registers contain sensitive timing parameters and settings for the DDR SDRAM. These registers are programmed with values that are in the operating range of the DDR used.

Writing to reserved fields or writing any reserved values in register bits can cause the dynamic memory controller to function erroneously.

PHY DLL Calibration

The PHY DLL calibration is performed as part of the SDRAM power-up initialization. It calibrates data against the DQS and CLK signal. However, running DLL calibration after self-refresh or at an arbitrary time is required in certain cases.

The DMC allows PHY DLL calibration to start by setting the `DMC_CTL.DLLCAL` bit. The `DMC_STAT.DLLCALDONE` bit can be used to monitor the progress of the calibration. Once calibration is over, this bit is set. Once the calibration procedure is started by writing to the `DMC_CAL_PADCTL0.CALSTRT` bit, the full calibration takes 300 DCLK cycles to complete.

NOTE: DLL calibration can be initiated only when the DMC is idle (`DMC_STAT.IDLE= 1`).

DDR3 ZQ Calibration Short CMD

The ZQ calibration short command is generally used to correct small variations in ZQ (-0.5%). To perform ZQ calibration, the controller is first checked for its idle state. Once the idle bit is obtained, a ZQCS command can be issued by setting the `DMC_STAT.ZQCSDONE` bit (0x0004). The `DMC_STAT.ZQCSDONE` bit (0x0008) can be used to monitor the calibration sequence. When this bit is 0, it indicates that the calibration is ongoing. When it is 1, it indicates that the calibration is done. As an example, a GP timer can be used to periodically trigger a ZQCS command to address tiny variations.

NOTE: The ZQ calibration function is essential for normal operation of DDR3. With the reference of the external resistance ($240 \Omega \pm 1\%$) connected to the `DMC_RZQ` pin, DDR3 calibrates the R_{on} and R_{tt} values of the ZQ pin against temperature and voltage variations.

DDR3 ZQ Calibration Long CMD

Several DDR3 impedance calibrations are implemented for optimal signal integrity. The long ZQ calibration is used after power-up and the short ZQ calibration is used periodically during normal operation to compensate for voltage and temperature drift. These calibration sequences improve connectivity between the SDRAM pads and the PCB trace. The `DMC_RZQ` pin on the SDRAM is connected to an external precision resistor that adjusts the output driver impedance R_{tt} and ODT values to match the trace impedance. The connection reduces impedance discontinuity and minimizes signal reflections.

The command has two variants named as ZQ calibration long (ZQCL) and ZQ calibration short (ZQCS). The ZQCL command is issued during initialization and after self-refresh exit command. It can be issued later depending on the system environment.

The DMC pads can be autocalibrated to the required driver impedance R_{tt} using an external resistance RZQ and the On Die Termination (ODT) value using the corresponding bits (`DMC_CAL_PADCTL2`). The autocalibration

logic translates these values into a corresponding drive strength control inside the PHY and then routed to the PADS. Autocalibration starts as soon as the bit is programmed (set the `DCLK` at the required frequency before setting this `DMC_CAL_PADCTL0.CALSTRT` bit). Autocalibration expects the program to select two different member sets of pads (address/command pads versus CLK/Data/DQS/DM pads).

Leveling Techniques

The DMC controller supports read and write leveling.

Write Leveling

It is difficult to meet the specifications with the on-board parasitic, trace length variability along with high frequency. The DMC controller supports leveling to compensate the skews between clock and strobe. This skew compensation is done independently for every byte lane. Write leveling is performed by the DDR3 memory controller as a part of initialization by programming the `DMC_MR1.WL` bit.

To dynamically perform write leveling:

1. Ensure that the DMC is idle.
2. Program the `DMC_MR1.WL` bit.
3. Enable the `DMC_MSK` register for the `DMC_MR1` register.
4. Wait for 2000 DMC clock cycles to complete write leveling.

Read Leveling

Read leveling is useful for calibrating read data timing. It is compensated for imbalanced loading on read path. Read leveling can be performed on LSB or byte lane. It is performed as a part of initialization sequence by setting the `DMC_EMR3.MPR` bit.

To dynamically perform read leveling:

1. Ensure that the DMC is idle.
2. Program the `DMC_EMR3.MPR` bit.
3. Enable DMC mask register `DMC_EMR3` register.
4. Wait for 2000 DMC clock cycles to complete read leveling.

Read Leveling during DQS Strobe Gating

To ensure robust read timing, read leveling must be done individually for all data bits. Some DDR3 memory devices do not drive read leveling patterns on all data pins. The devices send pattern only information on the LSBs of each byte lane. In these cases, the `DMC_EMR3` register cannot be used for read leveling. Instead, use the `DMC_CTL.RL_DQS` bit for read leveling.

Read leveling during DQS strobe gating is dynamically supported except during initialization.

Initializing the DMC

Complete the following procedures to initialize the DMC module.

- [Resetting the DMC Lane](#)
- [Performing ZQ Calibration](#)
- [Programming the DMC Controller](#)

Resetting the DMC Lane

If there is a change in the DMC clock frequency, the DDR lane must be reset using the following procedure. Once the lane is reset, wait 9000 DDR clock cycles for the DDR DLL to lock before restarting the DDR.

1. Set the `DMC_DDR_LANE0_CTL0.CB_RSTDLL` and `DMC_DDR_LANE1_CTL0.CB_RSTDLL` bits.
2. Change the DMC clock frequency.
3. Clear the `DMC_DDR_LANE0_CTL0.CB_RSTDLL` and `DMC_DDR_LANE1_CTL0.CB_RSTDLL` bits.

Performing ZQ Calibration

Perform the following ZQ calibration procedure for the proper impedance matching.

1. Program the `DMC_DDR_ZQ_CTL0.IMPWRADD` bit field with the drive strength of the address and command signals. A 100Ω drive strength is recommended for the drive strength of address and command signals. When programming a drive strength of 100Ω, write 0x64 into this bit field.
2. Program the `DMC_DDR_ZQ_CTL0.IMPWRDQ` bit field with the drive strength of the DQ, DQS, DM and clock signals. A 100Ω drive strength is recommended for the drive strength of address and command signals. When programming a drive strength of 100Ω, write 0x64 into this bit field.
3. Program the `DMC_DDR_ZQ_CTL0.IMPRTT` bit field with the adjusted On Die Termination (ODT) for the Data and DQS signals for the read operation. Due to a correction factor, program this field to 80% of the equivalent ODT.

$$\text{DMC_DDR_ZQ_CTL0.IMPRTT value} = \text{ODT} * 2 * 0.8$$

For example, if a 50Ω terminating resistance is required on the data pads to match the trace impedance to the board impedance, there will be two 50Ω resistance data pads in parallel. The value is programmed to $100 \times 0.8 = 80$.

4. Write 0 to the `DMC_DDR_ZQ_CTL1` register.
5. Program 0x70000000 to the `DMC_DDR_ZQ_CTL2` register.
6. Program 0x00000000 to the `DMC_DDR_CA_CTL` register.
7. Program 0x00000000 to the `DMC_DDR_ROOT_CTL` register.
8. Program 0x00010000 to the `DMC_DDR_ROOT_CTL` register.

9. Wait for 8000 DMC clock cycles.
10. Program 0x0C000001 to the `DMC_DDR_CA_CTL` register.
11. Wait for 8000 DMC clock cycles.
12. Program 0x00000000 to the `DMC_DDR_CA_CTL` register.
13. Program 0x00000000 to the `DMC_DDR_ROOT_CTL` register.

Programming the DMC Controller

Perform the following procedure to program the DMC controller.

1. Program the `DMC_CFG` register with the size of the DDR memory device. Program the `DMC_CFG.IFWID` and `DMC_CFG.SDRWID` bit fields to 16-bits wide.
2. Program the `DMC_TR0`, `DMC_TR1`, `DMC_TR2` registers with the timing parameters as required by the DDR memory device.
3. Program the `DMC_MR` and `DMC_MR1` registers with the appropriate values. A memory ODT of 120Ω is recommended. Write leveling is recommended for a 666 MHz operating frequency.
4. Program the `DMC_MR2` register. If read leveling is required, program the `DMC_EMR3` register. Read leveling is recommended for a 666 MHz operating frequency.
5. Program the `DMC_DLLCTL` register. Program the `DMC_DLLCTL.DATACYC` bit field with 15ns/ddr clock period in ns. Program the `DMC_DLLCTL.DLLCALRDCNT` bit field with 240.
6. Wait for 2000 DMC clock cycles.
7. Set the `DMC_DDR_CA_CTL.SW_REFRESH` bit, keeping the other bits unchanged.
8. Wait for five DMC clock cycles.
9. Set the `DMC_DDR_ROOT_CTL.SW_REFRESH` bit. Program the `DMC_DDR_ROOT_CTL.PIPE_OFSTDCYCLE` bit field with 2.
10. Program the `DMC_CTL` register.
 - `DMC_CTL.RDTOWR = 5`
 - Set the `DMC_CTL.DDR3EN` bit.
 - Set `DMC_CTL.AL_EN` for 666 MHz.
 - Set `DMC_CTL.RL_DQS` if operating frequency is great than 666 MH and to enable read leveling during data strobing.
 - Set the `DMC_CTL.INIT` bit.
11. If write leveling is enabled, wait for 600 DMC clock cycles or until the `DMC_MR1.WL` bit auto clears.
12. If read leveling is enabled wait for 2000 DMC clock cycles or until the `DMC_EMR3.MPR` bit auto clears.

13. If read leveling during data strobe gating is enabled, wait for 600 DMC clock cycles or until the `DMC_CTL.RL_DQS` bit auto clears.
14. Wait for 722000 DMC clock cycles to complete DMC initialization or until the `DMC_STAT.INITDONE` bit is set.
15. Set the `DMC_DDR_LANE0_CTL1.COMP_DCYCLE` and `DMC_DDR_LANE1_CTL1.COMP_DCYCLE` bits, without affecting other bits.
16. Wait for 10 DMC clock cycles.
17. Clear the `DMC_DDR_LANE0_CTL1.COMP_DCYCLE` and `DMC_DDR_LANE1_CTL1.COMP_DCYCLE` bits, without affecting other bits.
18. Set and then clear the `DMC_DDR_LANE0_CTL0.CB_RSTDAT` bit, without affecting other bits.
19. Set and then clear `DMC_DDR_LANE1_CTL0.CB_RSTDAT` bit, without affecting other bits.
20. Wait for 2500 DMC clock cycles.
21. Read the `DMC_STAT.PHYRDPHASE` bit field. Program this value into the `DMC_DLLCTL.DATACYC` bit field, without affecting other bits in the `DMC_DLLCTL` register.
22. Reprogram the `DMC_CTL.DDR3EN` and `DMC_CTL.RDTOWR` bits. If the operation frequency is 666 MHz set the `DMC_CTL.AL_EN` bit.

ADSP-2156x DMC Register Descriptions

Dynamic Memory Controller (DMC) contains the following registers.

Table 9-5: ADSP-2156x DMC Register List

Name	Description
<code>DMC_CFG</code>	Configuration Register
<code>DMC_CTL</code>	Control Register
<code>DMC_DLLCTL</code>	DLL Control Register
<code>DMC_DT_CALIB_ADDR</code>	Data Calibration Address Register
<code>DMC_DT_DATA_CALIB_DATA0</code>	Data Calibration Data 0 Register
<code>DMC_DT_DATA_CALIB_DATA1</code>	Data Calibration Data 1 Register
<code>DMC_EFFCTL</code>	Efficiency Control Register
<code>DMC_EMR3</code>	Shadow EMR3 Register
<code>DMC_MR</code>	Shadow MR0 Register (DDR3)
<code>DMC_MR1</code>	Shadow MR1 Register (DDR3)
<code>DMC_MR2</code>	Shadow MR2 Register (DDR3)

Table 9-5: ADSP-2156x DMC Register List (Continued)

Name	Description
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

Configuration Register

The `DMC_CFG` register selects SDRAM device specific parameters and selects the SDRAM interface width.

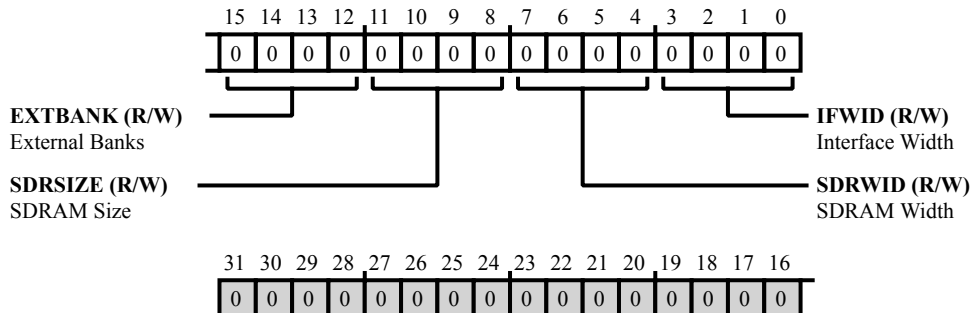


Figure 9-1: DMC_CFG Register Diagram

Table 9-6: DMC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:12 (R/W)	EXTBANK	External Banks. The <code>DMC_CFG</code> . <code>EXTBANK</code> bits select the number of external banks connected to the DMC. Note that all values other than those shown are reserved.
		0 1 External Bank
		1-15 Reserved
11:8 (R/W)	SDRSIZE	SDRAM Size. The <code>DMC_CFG</code> . <code>SDRSIZE</code> bits select the size of individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.
		3 512M Bit SDRAM
		4 1G Bit SDRAM
		5 2G Bit SDRAM
		6 4G Bit SDRAM
7:4 (R/W)	SDRWID	SDRAM Width. The <code>DMC_CFG</code> . <code>SDRWID</code> bits select the width of the individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.
		0-1 Reserved
		2 16-Bit Wide SDRAM
		3-15 Reserved

Table 9-6: DMC_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	IFWID	Interface Width. The DMC_CFG.IFWID bits select the width of the interface between the DMC and SDRAM. Note that all values other than those shown are reserved.	
		0-1	Reserved
		2	16-Bit Wide Interface. All other values are reserved. This field specifies the interface width between the controller and the SDRAM.
		3-15	Reserved

Control Register

The `DMC_CTL` register controls DMC modes, DLL calibration, and DRAM initialization.

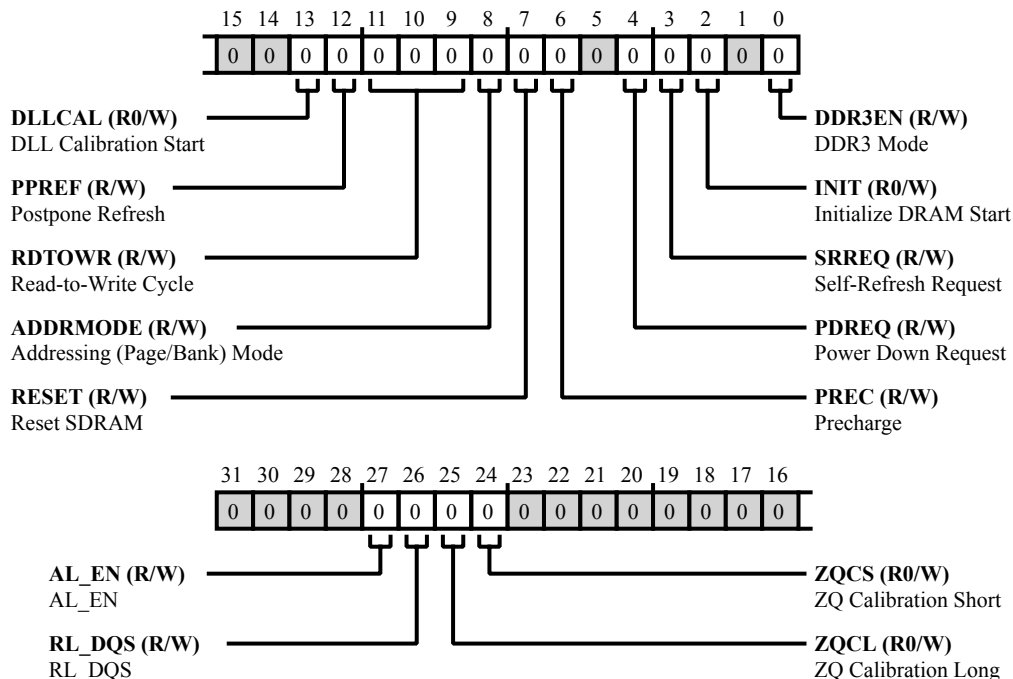


Figure 9-2: `DMC_CTL` Register Diagram

Table 9-7: `DMC_CTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27 (R/W)	AL_EN	AL_EN. When set, enables 800MHz operation
		0 Disables greater than 667MHz operation
		1 Enables greater than 667MHz operation
26 (R/W)	RL_DQS	RL_DQS. When set, enables Read leveling during DQS Gating Training. This bit auto-clears on read leveling completion
		0 Enables Read leveling during DQS Gating Training
		1 Disables Read leveling during DQS Gating training

Table 9-7: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R0/W)	ZQCL	ZQ Calibration Long. The DMC_CTL . ZQCL bit starts the ZQ calibration long sequence. Note that this bit always reads as 0.
		0 No effect
		1 Triggers ZQ calibration long sequence
24 (R0/W)	ZQCS	ZQ Calibration Short. The DMC_CTL . ZQCS bit starts the ZQ calibration short sequence. Note that this bit always reads as 0.
		0 No effect
		1 Triggers ZQ calibration short sequence
13 (R0/W)	DLLCAL	DLL Calibration Start. The DMC_CTL . DLLCAL bit starts the PHY DLL calibration sequence. Note that this bit always reads as 0.
		0 No effect
		1 Start PHY DLL calibration
12 (R/W)	PPREF	Postpone Refresh. The DMC_CTL . PPREF bit enables postponing the DMCs sending of auto-refresh commands. When enabled, the DMC accumulates refresh commands. The DMC_EFFCTL . NUMREF field selects the number of refresh commands that the DMC may accumulate. When disabled, the DMC_TR1 . TREF field selects the interval for auto-refresh command distribution. A maximum of eight auto-refresh commands can be accumulated in DDR3 mode.
		0 Disable Postpone Refresh
		1 Enable Postpone Refresh
11:9 (R/W)	RDTOWR	Read-to-Write Cycle. The DMC_CTL . RDTOWR bits select the number of cycles that the DMC adds when a write operation follows a read operation. For proper operation, it should be programmed with the value of 010.
		0 1 Cycle Added from JEDEC Spec Value
		1 2 Cycles Added from JEDEC Spec Value
		2 3 Cycles Added from JEDEC Spec Value
		3 4 Cycles Added from JEDEC Spec Value
		4 5 Cycles Added from JEDEC Spec Value
		5 6 Cycles Added from JEDEC Spec Value

Table 9-7: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		6 7 Cycles Added from JEDEC Spec Value
		7 8 Cycles Added from JEDEC Spec Value
8 (R/W)	ADDRMODE	Addressing (Page/Bank) Mode. The DMC_CTL.ADDRMODE bit selects whether the DMC uses page or bank interleaving for addressing. When using page interleaving, the bank address bits follow the most significant column address bits. When using bank interleaving, the bank address bits follow the most significant row address bits.
		0 Bank Interleaving
		1 Page Interleaving
7 (R/W)	RESET	Reset SDRAM. The DMC_CTL.RESET bit starts the reset sequence. Note that this bit always reads as 0.
		0 No effect
		1 Starts reset sequence
6 (R/W)	PREC	Precharge. The DMC_CTL.PREC bit enables precharge, which closes DRAM rows immediately after access. When disabled, all accesses result in the respective DRAM rows remaining open, until the DMC needs to close them.
		0 No Effect
		1 Enable Precharge
4 (R/W)	PDREQ	Power Down Request. The DMC_CTL.PDREQ bit enables power-down mode. When the DMC is in power-down mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in power-down mode as long as this bit is 1.
		0 Disable Power-Down
		1 Enable Power-Down
3 (R/W)	SRREQ	Self-Refresh Request. The DMC_CTL.SRREQ bit enables self-refresh mode. When the DMC is in self-refresh mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in self-refresh mode as long as this bit is 1.
		0 Disable Self-Refresh
		1 Enable Self-Refresh

Table 9-7: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R0/W)	INIT	Initialize DRAM Start. The DMC_CTL . INIT bit starts the power up DRAM initialization sequence and DLL calibration sequence. Note that this bit always reads as 0.
		0 No Effect
		1 Start DRAM Initialization
0 (R/W)	DDR3EN	DDR3 Mode. The DMC_CTL . DDR3EN bit selects whether the DMC operates in DDR3 mode.
		0 Reserved
		1 Enable DDR3 mode

DLL Control Register

The `DMC_DLLCTL` register holds the programmable parameters associated with the DLLs within the DMC PHY.

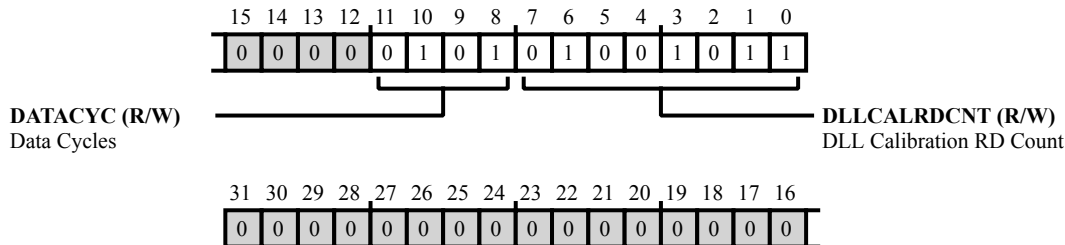


Figure 9-3: DMC_DLLCTL Register Diagram

Table 9-8: DMC_DLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	DATACYC	Data Cycles. The <code>DMC_DLLCTL.DATACYC</code> bits select the latency after which the DMC reads data from the PHY. This field must be written with the value indicated in the <code>DMC_STAT.PHYRDPHASE</code> field, or data corruption occurs on all SDRAM reads.
		2 2 Clock Cycles Latency
		3 3 Clock Cycles Latency
		4 4 Clock Cycles Latency
		5 5 Clock Cycles Latency
		6 6 Clock Cycles Latency
		7 7 Clock Cycles Latency
		8 8 Clock Cycles Latency
		9 9 Clock Cycles Latency
		10 10 Clock Cycles Latency
		11 11 Clock Cycles Latency
		12 12 Clock Cycles Latency
		13 13 Clock Cycles Latency
		14 14 Clock Cycles Latency
15 15 Clock Cycles Latency		
7:0 (R/W)	DLLCALRDCNT	DLL Calibration RD Count. The <code>DMC_DLLCTL.DLLCALRDCNT</code> field selects the number of read operations that the PHY uses for DLL calibration.

Data Calibration Address Register

The `DMC_DT_CALIB_ADDR` register provides the address used for the data calibration for read and write. During the DMC PHY DLL calibration, a particular set of locations in the DRAM is written and a series of reads are performed back to back to calibrate the PHY. The DMC PHY needs prior information about the data that would be read during the PHY DLL calibration. The controller performs one burst write operation to the address programmed in `DMC_DT_CALIB_ADDR` (0x0090).

Note: While the exact address chosen does not matter much during memory initialization, if calibration of the PHY is performed when the DRAM contains valid data, care needs to be taken to ensure that this address points to an unused address. Else, this operation will modify application data stored at the address selected.

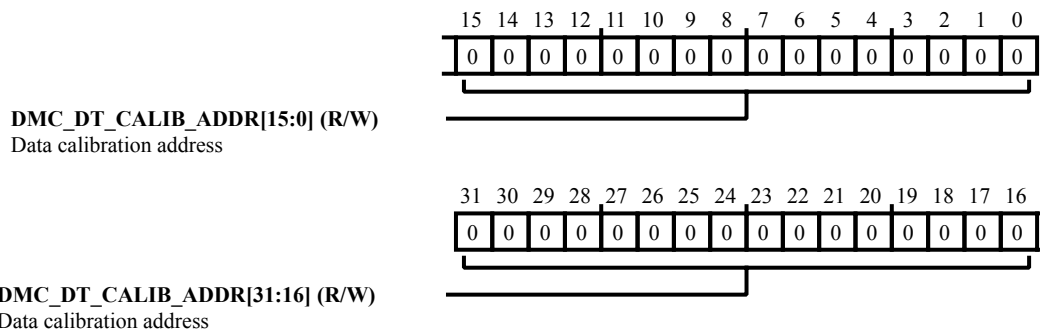


Figure 9-4: DMC_DT_CALIB_ADDR Register Diagram

Table 9-9: DMC_DT_CALIB_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DMC_DT_CALIB_ADDR	Data calibration address. The <code>DMC_DT_CALIB_ADDR.DMC_DT_CALIB_ADDR</code> bit field contains the address to be programmed for the data calibration for read and write.

Data Calibration Data 0 Register

The `DMC_DT_DATA_CALIB_DATA0` register contains the first 32-bit data used for the write during the data calibration.

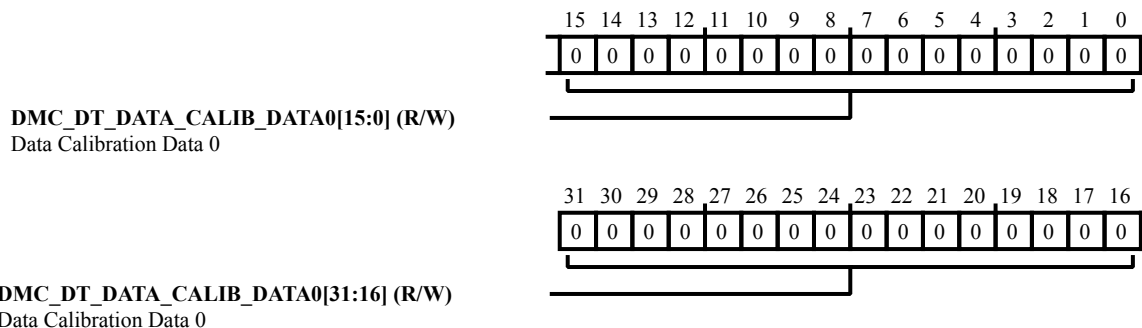


Figure 9-5: `DMC_DT_DATA_CALIB_DATA0` Register Diagram

Table 9-10: `DMC_DT_DATA_CALIB_DATA0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	<code>DMC_DT_DATA_CALIB_DATA0</code>	Data Calibration Data 0. The <code>DMC_DT_DATA_CALIB_DATA0.DMC_DT_DATA_CALIB_DATA0</code> bit field contains the first 32 bit data used for the write during the data calibration.

Data Calibration Data 1 Register

The `DMC_DT_DATA_CALIB_DATA1` register contains the second 32-bit data used for the write during the data calibration.

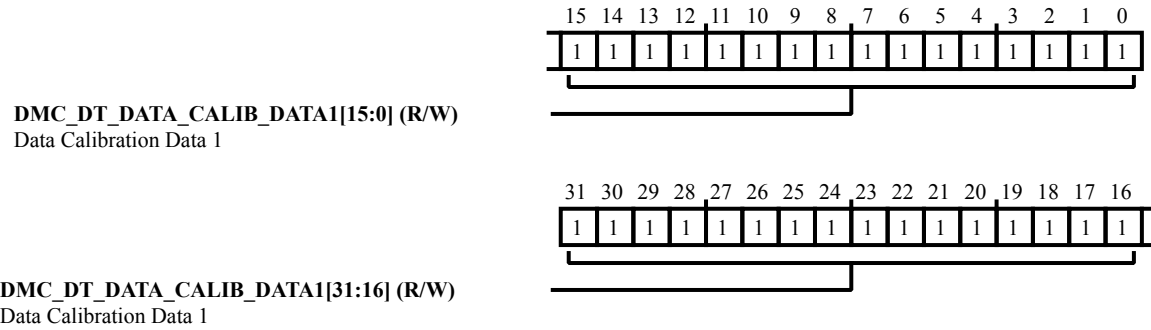


Figure 9-6: `DMC_DT_DATA_CALIB_DATA1` Register Diagram

Table 9-11: `DMC_DT_DATA_CALIB_DATA1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	<code>DMC_DT_DATA_CAL- IB_DATA1</code>	Data Calibration Data 1. The <code>DMC_DT_DATA_CALIB_DATA1.DMC_DT_DATA_CALIB_DATA1</code> bit field contains the second 32 bit data used for the write during the data calibration.

Efficiency Control Register

The `DMC_EFFCTL` register control DMC features that improve throughput efficiency. These include features such as auto-refresh management, precharge options, and write data options.

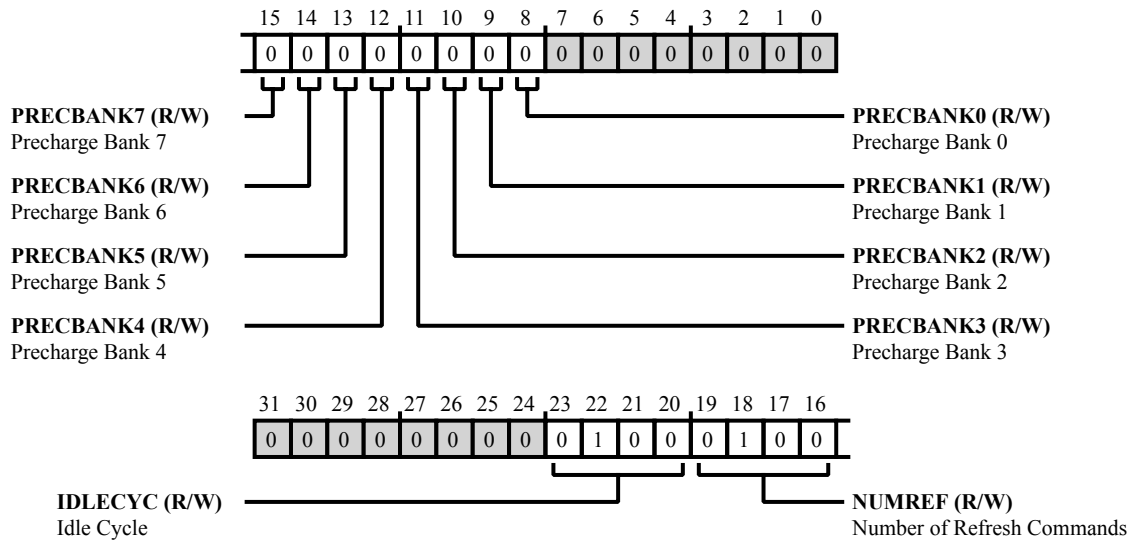


Figure 9-7: DMC_EFFCTL Register Diagram

Table 9-12: DMC_EFFCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:20 (R/W)	IDLECYC	<p>Idle Cycle.</p> <p>The <code>DMC_EFFCTL.IDLECYC</code> bits select the number of cycles after which the DMC issues any accumulated auto-refresh commands if postpone refresh is enabled (<code>DMC_CTL.PPREF = 1</code>). When <code>DMC_EFFCTL.IDLECYC</code> is set to 0, the DMC ignores the <code>DMC_CTL.PPREF</code> selection and does not accumulate/postpone periodic auto-refresh commands.</p> <p>Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles.</p> <p>Note 2: This value is ignored if <code>DMC_CTL.PPREF</code> is not set.</p> <p>Note 3: Setting this value to 0000 overrides the "postpone refresh" feature and does not accumulate/postpone periodic auto refreshes.</p>
		0-15 0 to 15 Idle Cycles to Postpone Refresh Commands

Table 9-12: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/W)	NUMREF	Number of Refresh Commands. The DMC_EFFCTL.NUMREF bits select the number of auto-refresh commands that the DMC can accumulate if postpone refresh is enabled (DMC_CTL.PPREF =1). In DDR3 mode, the DMC may accumulate up to eight auto-refresh commands. Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles. Note 2: This value is ignored if DMC_CTL.PPREF is not set.
		0 No Refresh Commands Accumulate
		1 1 Refresh Command May Accumulate
		2 2 Refresh Commands May Accumulate
		3 3 Refresh Commands May Accumulate
		4 4 Refresh Commands May Accumulate
		5 5 Refresh Commands May Accumulate
		6 6 Refresh Commands May Accumulate
		7 7 Refresh Commands May Accumulate
8 8 Refresh Commands May Accumulate		
15 (R/W)	PRECBANK7	Precharge Bank 7. The DMC_EFFCTL.PRECBANK7 bit enables precharge (closes the page) of bank 7 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 7
		1 Enable Precharge Bank 7
14 (R/W)	PRECBANK6	Precharge Bank 6. The DMC_EFFCTL.PRECBANK6 bit enables precharge (closes the page) of bank 6 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 6
		1 Enable Precharge Bank 6

Table 9-12: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	PRECBANK5	Precharge Bank 5. The DMC_EFFCTL.PRECBANK5 bit enables precharge (closes the page) of bank 5 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 5
		1 Enable Precharge Bank 5
12 (R/W)	PRECBANK4	Precharge Bank 4. The DMC_EFFCTL.PRECBANK4 bit enables precharge (closes the page) of bank 4 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 4
		1 Enable Precharge Bank 4
11 (R/W)	PRECBANK3	Precharge Bank 3. The DMC_EFFCTL.PRECBANK3 bit enables precharge (closes the page) of bank 3 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 3
		1 Enable Precharge Bank 3
10 (R/W)	PRECBANK2	Precharge Bank 2. The DMC_EFFCTL.PRECBANK2 bit enables precharge (closes the page) of bank 2 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 2
		1 Enable Precharge Bank 2
9 (R/W)	PRECBANK1	Precharge Bank 1. The DMC_EFFCTL.PRECBANK1 bit enables precharge (closes the page) of bank 1 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.
		0 Disable Precharge Bank 1
		1 Enable Precharge Bank 1

Table 9-12: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PRECBANK0	Precharge Bank 0. The DMC_EFFCTL.PRECBANK0 bit enables precharge (closes the page) of bank 0 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 0
		1	Enable Precharge Bank 0

Shadow EMR3 Register

The `DMC_EMR3` register in the DMC shadows the EMR3 register in the SDRAM when the DMC is in DDR3 mode (`DMC_CTL.DDR3EN = 1`). If unmasked by the corresponding bit in the shadow mask register (`DMC_MSK.EMR3 = 1`), a write to `DMC_EMR3` triggers an extended “mode register set” command on the memory interface. If masked, a write to `DMC_EMR3` only updates the register in the DMC, not the register in the SDRAM.

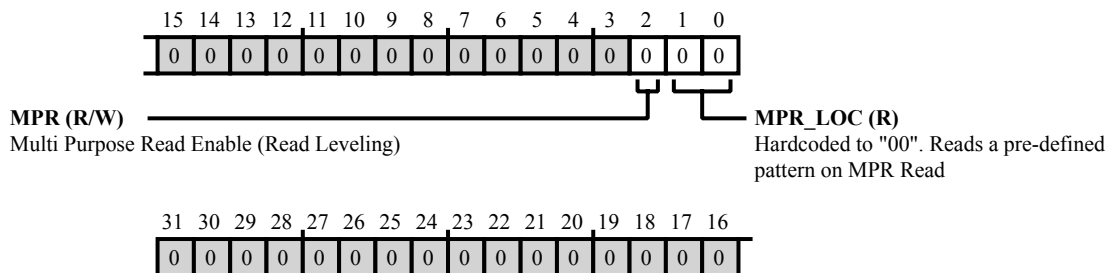


Figure 9-8: DMC_EMR3 Register Diagram

Table 9-13: DMC_EMR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	MPR	Multi Purpose Read Enable (Read Leveling). The <code>DMC_EMR3.MPR</code> bit enables high temperature self-refresh rate.
		0 Disable
		1 Enable
1:0 (R/NW)	MPR_LOC	Hardcoded to "00". Reads a pre-defined pattern on MPR Read.

Shadow MR0 Register (DDR3)

The `DMC_MR` register in the DMC shadows the MR register in the SDRAM when the DMC is DDR3 mode (`DMC_CTL.DDR3EN=1`). If unmasked by the corresponding bit in the shadow mask register (`DMC_MSK.MR=1`), a write to `DMC_MR` triggers a “mode register set” command on the memory interface. If masked, a write to `DMC_MR` only updates the register in the DMC, not the register in the SDRAM.

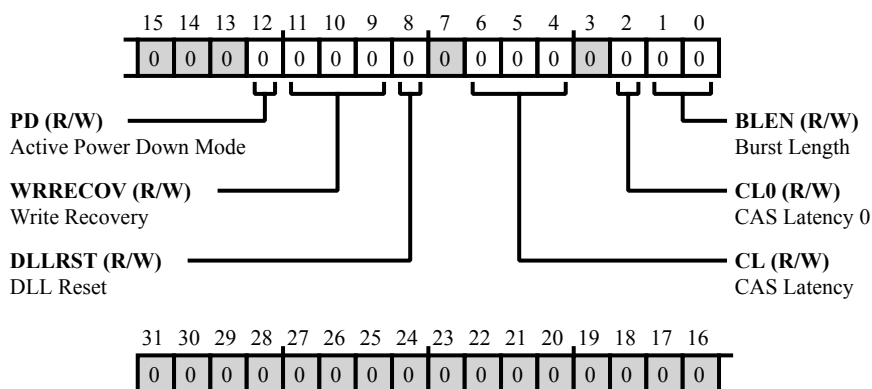


Figure 9-9: DMC_MR Register Diagram

Table 9-14: DMC_MR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	PD	Active Power Down Mode. The <code>DMC_MR.PD</code> bit selects the active power-down mode. Note that this parameter applies only for DDR3 mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.
		0 Fast Exit (normal)
		1 Slow Exit (low power)
11:9 (R/W)	WRRECOV	Write Recovery. The <code>DMC_MR.WRRECOV</code> bit selects the write recovery time in terms of clock cycles (t_{CK}). Note that this parameter applies only for DDR3 mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.
		0 16 clock cycles for DDR3
		1 5 clock cycles for DDR3
		2 6 clock cycles for DDR3
		3 7 clock cycles for DDR3
		4 8 clock cycles for DDR3
		5 10 clock cycles for DDR3
		6 12 clock cycles for DDR3

Table 9-14: DMC_MR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		7 14 clock cycles for DDR3
8 (R/W)	DLLRST	<p>DLL Reset.</p> <p>The DMC_MR.DLLRST bit initiates a DLL reset on the SDRAM. Note that this parameter applies only for DDR3 mode. For more information about this operation, see the data sheet for the SDRAM being used in your system.</p>
		0 Normal Operation
		1 Reset DLL
6:4 (R/W)	CL	<p>CAS Latency.</p> <p>The DMC_MR.CL bit field select latency from the assertion of a read/write signal to the SDRAM until the first valid data on the output from the SDRAM in terms of clock cycles. For more information about this operation, see the data sheet for the SDRAM being used in your system.</p> <p>For DDR3 only, Bit[2] of this register must be used along with [6:4]. Following CAS values are seen.</p> <p>"0010": 5 "0100": 6 "0110": 7 "1000": 8 "1010": 9 "1100": 10 "1110": 11 "0001": 12 "0011": 13 "0101": 14 All other combinations are reserved.</p>
2 (R/W)	CL0	<p>CAS Latency 0.</p> <p>The DMC_MR.CL0 bit is applicable for DDR3 only and is used in conjunction with the DMC_MR.CL bits.</p>
1:0 (R/W)	BLLEN	<p>Burst Length.</p> <p>The DMC_MR.BLEN bits select burst length for transfers. For more information about this operation, see the data sheet for the SDRAM being used in your system. Note that values other than those shown are not supported.</p>
		0 8-Bit Burst Length - DDR3 only
		2 Reserved
		3 Reserved

Shadow MR1 Register (DDR3)

The `DMC_MR1` register is a mirror of the DDR3 SDRAM Mode register 1. This register is used only when the DMC is operating in DDR3 mode. A write to this register triggers an extended "mode register 1 set" command on the memory interface provided the corresponding mask bit is set in the mask register. Else, only the mirror register is updated.

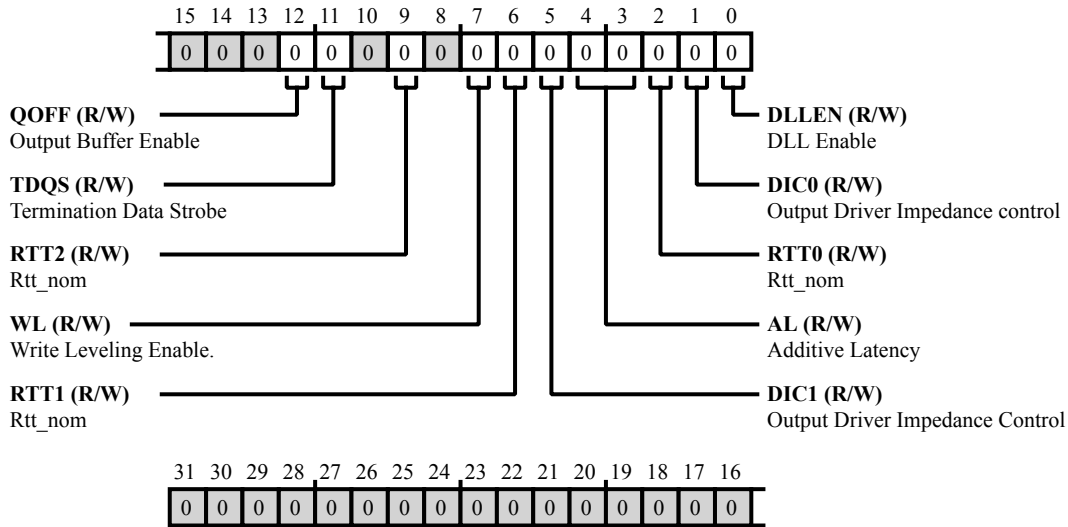


Figure 9-10: DMC_MR1 Register Diagram

Table 9-15: DMC_MR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	QOFF	Output Buffer Enable. The <code>DMC_MR1.QOFF</code> bit enables the SDRAM output pins. For more information about this operation, see the data sheet for the SDRAM being used in your system.
		0 Output buffer enabled
		1 Output buffer disabled
11 (R/W)	TDQS	Termination Data Strobe. The <code>DMC_MR1.TDQS</code> bit provides additional termination resistance outputs that may be useful in some system configurations. The <code>DMC_MR1.TDQS</code> bit is not supported in x4 or x16 configurations. When enabled via the mode register, the same termination resistance function is applied to the TDQS/TDQS# pins that is applied to the DQS/DQS# pins.
		0 Enable
		1 Disable

Table 9-15: DMC_MR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration				
9 (R/W)	RTT2	<p>Rtt_nom.</p> <p>The DMC_MR1 . RTT2 bit is used in conjunction with the DMC_MR1 . RTT0 and DMC_MR1 . RTT1 bits.</p> <p>(9 6 2)</p> <p>0 0 0 Rtt_Nom disabled</p> <p>0 0 1 RZQ/4</p> <p>0 1 0 RZQ/2</p> <p>0 1 1 RZQ/6</p> <p>1 0 0 RZQ/12 (reserved if Rtt_Nom is used during writes)</p> <p>1 0 1 RZQ/8 (reserved if Rtt_Nom is used during writes)</p> <p>1 1 0 Reserved</p> <p>1 1 1 Reserved</p>				
7 (R/W)	WL	<p>Write Leveling Enable..</p> <p>The DMC_MR1 . WL bit enables the SDRAM output pins. For more information about this operation, see the data sheet for the SDRAM being used in your system.</p> <table border="1" data-bbox="620 1045 1521 1144"> <tr> <td>0</td> <td>Disable</td> </tr> <tr> <td>1</td> <td>Enable</td> </tr> </table>	0	Disable	1	Enable
0	Disable					
1	Enable					
6 (R/W)	RTT1	<p>Rtt_nom.</p> <p>The DMC_MR1 . RTT1 bit combines with the DMC_MR1 . RTT0 bit to set the termination resistance. See the DMC_MR1 . RTT2 and DMC_MR1 . RTT0 bit description for more information.</p>				
5 (R/W)	DIC1	<p>Output Driver Impedance Control.</p> <p>The DMC_MR1 . DIC1 bit is used in conjunction with the DMC_MR1 . DIC0 bit.</p> <p>(5, 1)</p> <p>0 0 RZQ/6</p> <p>0 1 RZQ/7</p> <p>1 0 Reserved</p> <p>1 1 Reserved</p>				

Table 9-15: DMC_MR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:3 (R/W)	AL	Additive Latency. The DMC_MR1.AL bits select a number of added latency time for CAS operations in terms of clock cycles (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system.
		0 AL disabled
		1 CL-1
		2 CL-2
		3 Reserved
2 (R/W)	RTT0	Rtt_nom. The DMC_MR1.RTT0 bit combines with the DMC_MR1.RTT1 and DMC_MR1.RTT1 bits to set the termination resistance. See the DMC_MR1.RTT1 and DMC_MR1.RTT2 bit descriptions for more information.
1 (R/W)	DIC0	Output Driver Impedance control. The DMC_MR1.DIC0 bit is used with the DMC_MR1.DIC1 bit.
0 (R/W)	DLLEN	DLL Enable. The DMC_MR1.DLLEN bit enables the DLL in the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system.
		0 Enable
		1 Disable

Shadow MR2 Register (DDR3)

The `DMC_MR2` register mirrors DDR3 SDRAM device Mode register 2 when the controller is operating in DDR3 mode. A write to this register triggers an extended "mode register set" command on the memory interface provided the corresponding mask bit is set in the mask register. Else, only the mirror register is updated.

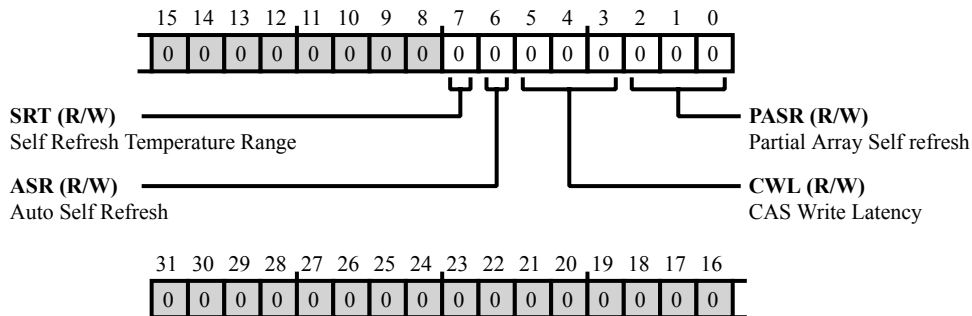


Figure 9-11: DMC_MR2 Register Diagram

Table 9-16: DMC_MR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	SRT	Self Refresh Temperature Range. The <code>DMC_MR2</code> . SRT bit enables high temperature self-refresh rate.
		0 Disable
		1 Enable
6 (R/W)	ASR	Auto Self Refresh.
		0 Manual SR Reference (SRT)
		1 ASR enable (Optional)
5:3 (R/W)	CWL	CAS Write Latency.
		0 5 clock cycles
		1 6 clock cycles
		2 7 clock cycles
		3 8 clock cycles
		4 9 clock cycles
		5 10 clock cycles
		6 11 clock cycles
7 12 clock cycles		

Table 9-16: DMC_MR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	PASR	Partial Array Self refresh. The DMC_MR2 . PASR bits select the amount of memory to be refreshed during self refresh. For more information about this operation, see the data sheet for the SDRAM being used in your system.
		0 4 banks: full array, 8 banks: full array
		1 4 banks: Half Array (BA[1:0]=00&01), 8 banks: Half Array (BA[2:0] = 000, 001, 010, &011)
		2 4 banks: Quarter Array (BA[1:0]=00), 8 banks: Quarter Array (BA[2:0] = 000&001)
		3 4 banks: not defined, 8 banks: 1/8th array (BA[2:0] = 000)
		4 4 banks: 3/4 Array (BA[1:0]=01, 10&11), 8 banks: 3/4 Array (BA[2:0] = 010, 011, 100, 101, 110, &111)
		5 4 banks: Half Array (BA[1:0]=10&11), 8 banks: Half Array (BA[2:0] = 100, 101, 110, &111)
		6 4 banks: Quarter Array (BA[1:0]=11), 8 banks: Quarter Array (BA[2:0] =110 & 111)
		7 4 banks: not defined, 8 banks: 1/8th array (BA[2:0] = 111)

Mask (Mode Register Shadow) Register

The `DMC_MSK` register permits masking (disabling) writes to the MR and EMRn registers in the SDRAM in DDR3 Mode. When masked, writes to these registers go instead to shadow copies of these registers (`DMC_MR`, `DMC_MR1`, `DMC_MR2`), which are maintained within the DMC. When a shadow register's corresponding bit is unmasked (enabled), the DMC generates the MRS or EMRS command to transfer the contents of the shadow register (in the DMC) to the actual register (in the SDRAM).

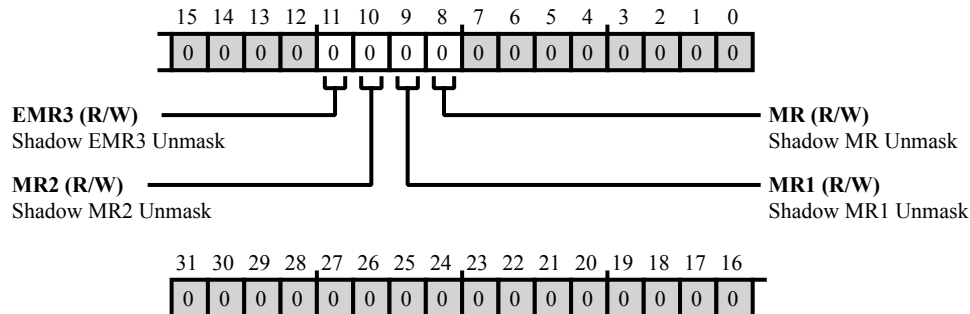


Figure 9-12: DMC_MSK Register Diagram

Table 9-17: DMC_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	EMR3	Shadow EMR3 Unmask. The <code>DMC_MSK</code> .EMR3 bit masks or unmasks writes to the EMR3 register in the SDRAM. When masked, writes to this register instead go to the EMR3 register. When unmasked, the DMC writes the EMR3 value to the EMR3 register in the SDRAM. After completing the write, the DMC clears this bit.
		0 Mask (Disable) Write to EMR3
		1 Unmask (Enable) Write to EMR3
10 (R/W)	MR2	Shadow MR2 Unmask. The <code>DMC_MSK</code> .MR2 bit masks or unmasks writes to the MR2 register (in DDR3 mode) in the SDRAM. When masked, writes to this register instead go to the <code>DMC_MR2</code> register. When unmasked, the DMC writes the <code>DMC_MR2</code> value to the MR2 register (in DDR3 mode) in the SDRAM. After completing the write, the DMC clears this bit.
		0 Mask (Disable) Write to EMR2
		1 Unmask (Enable) Write to EMR2

Table 9-17: DMC_MSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	MR1	Shadow MR1 Unmask. The <code>DMC_MSK.MR1</code> bit masks or unmasks writes to the MR1 register in the DDR3 SDRAM. When masked, writes to this register instead go to the <code>DMC_MR1</code> register. When unmasked, the DMC writes the <code>DMC_MR1</code> value to the MR1 register in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit is valid only for DDR3 Mode of operation.
		0 Mask (Disable) Write to EMR1
		1 Unmask (Enable) Write to EMR1
8 (R/W)	MR	Shadow MR Unmask. The <code>DMC_MSK.MR</code> bit masks or unmasks writes to the MR register in the SDRAM. When masked, writes to this register instead go to the <code>DMC_MR</code> register. When unmasked, the DMC writes the <code>DMC_MR</code> value to the MR register in the SDRAM. After completing the write, the DMC clears this bit.
		0 Mask (Disable) Write to MR
		1 Unmask (Enable) Write to MR

Priority ID Register 1

The `DMC_PRIO` register allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the `DMC_PRIOMSK` register to obtain a single SCB ID or a range of IDs that get elevated priority.

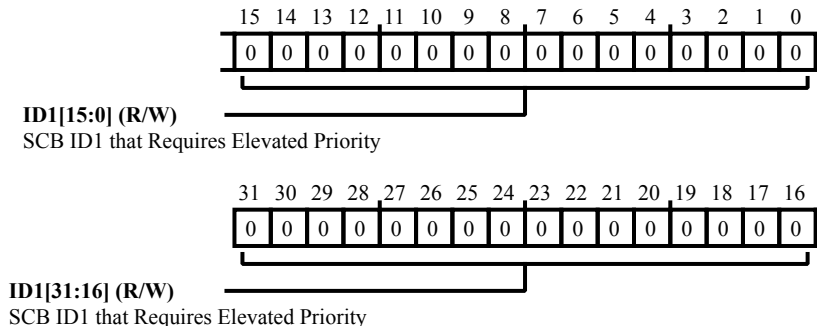


Figure 9-13: DMC_PRIO Register Diagram

Table 9-18: DMC_PRIO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1	SCB ID1 that Requires Elevated Priority.

Priority ID Register 2

The `DMC_PRIO2` register is another register which allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the `DMC_PRIOMSK2` register to obtain a single SCB ID or a range of IDs that get elevated priority.

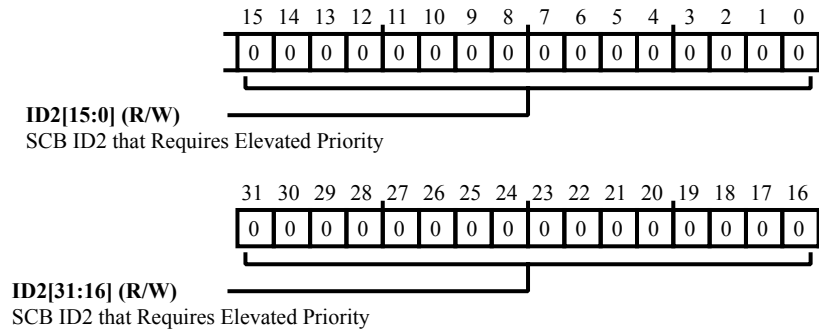


Figure 9-14: DMC_PRIO2 Register Diagram

Table 9-19: DMC_PRIO2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2	SCB ID2 that Requires Elevated Priority.

Priority ID Mask Register 1

The `DMC_PRIOMSK` register masks the respective ID bits in the `DMC_PRIOMSK` register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

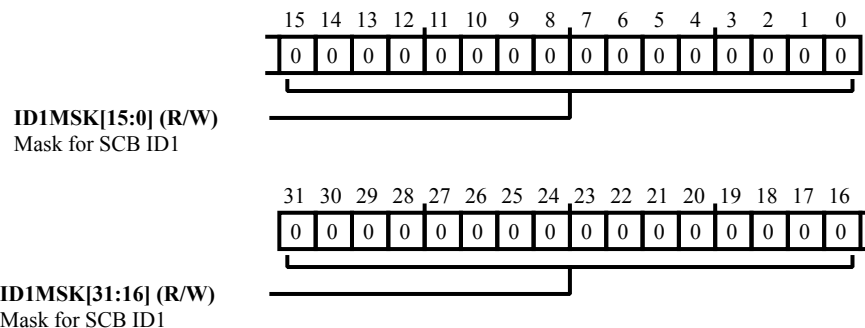


Figure 9-15: DMC_PRIOMSK Register Diagram

Table 9-20: DMC_PRIOMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1MSK	Mask for SCB ID1.

Priority ID Mask Register 2

The `DMC_PRIOMSK2` register bits mask the respective ID bits in the `DMC_PRIO2` register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

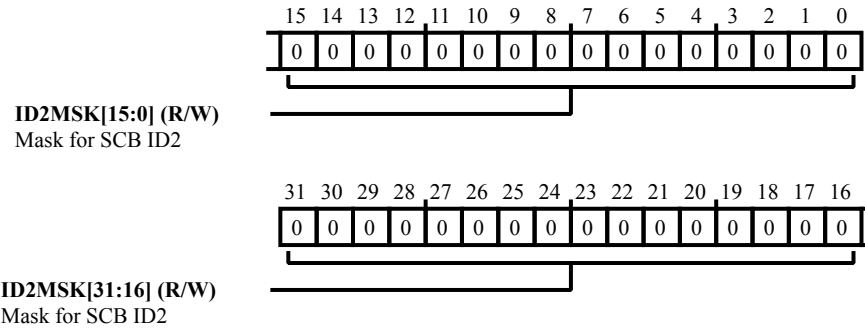


Figure 9-16: `DMC_PRIOMSK2` Register Diagram

Table 9-21: `DMC_PRIOMSK2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2MSK	Mask for SCB ID2.

DMC Read Data Buffer ID Register 1

The `DMC_RDDATABUFID1` register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the `DMC_RDDATABUFMSK1` register to obtain a single SCB ID or a range of IDs that get elevated priority.

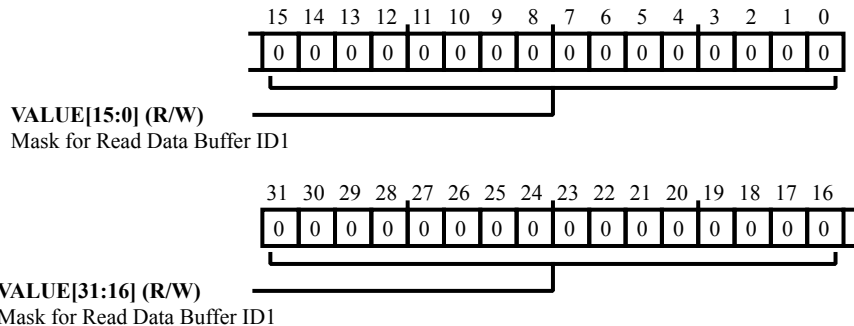


Figure 9-17: DMC_RDDATABUFID1 Register Diagram

Table 9-22: DMC_RDDATABUFID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer ID Register 2

The `DMC_RDDATABUFID2` register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the `DMC_RDDATABUFMSK2` register to obtain a single SCB ID or a range of IDs that get elevated priority.

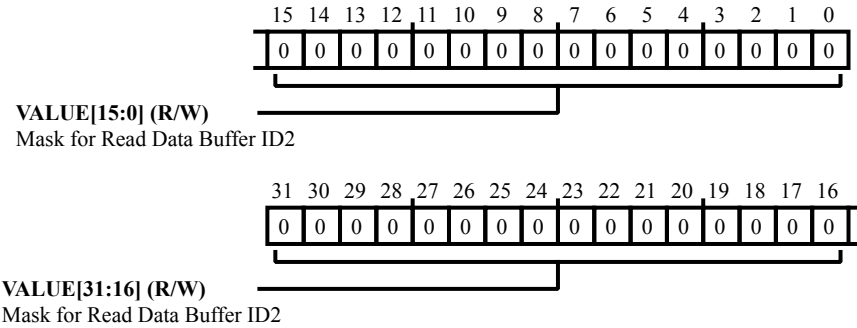


Figure 9-18: `DMC_RDDATABUFID2` Register Diagram

Table 9-23: `DMC_RDDATABUFID2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

DMC Read Data Buffer Mask Register 1

The `DMC_RDDATABUFMSK1` register bits mask the respective ID bits in the DMC Priority Mask ID register.

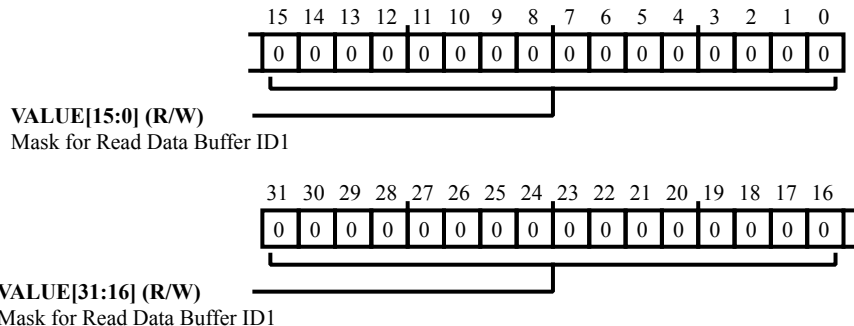


Figure 9-19: DMC_RDDATABUFMSK1 Register Diagram

Table 9-24: DMC_RDDATABUFMSK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer Mask Register 2

The `DMC_RDDATABUFMSK2` register bits mask the respective ID bits in the DMC Priority Mask ID register.

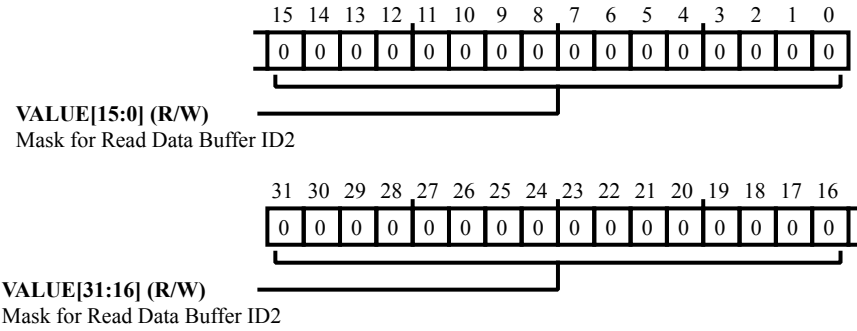


Figure 9-20: `DMC_RDDATABUFMSK2` Register Diagram

Table 9-25: `DMC_RDDATABUFMSK2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

Status Register

The `DMC_STAT` register indicates status for modes selected with the `DMC_CTL` register and indicates status DMC operations.

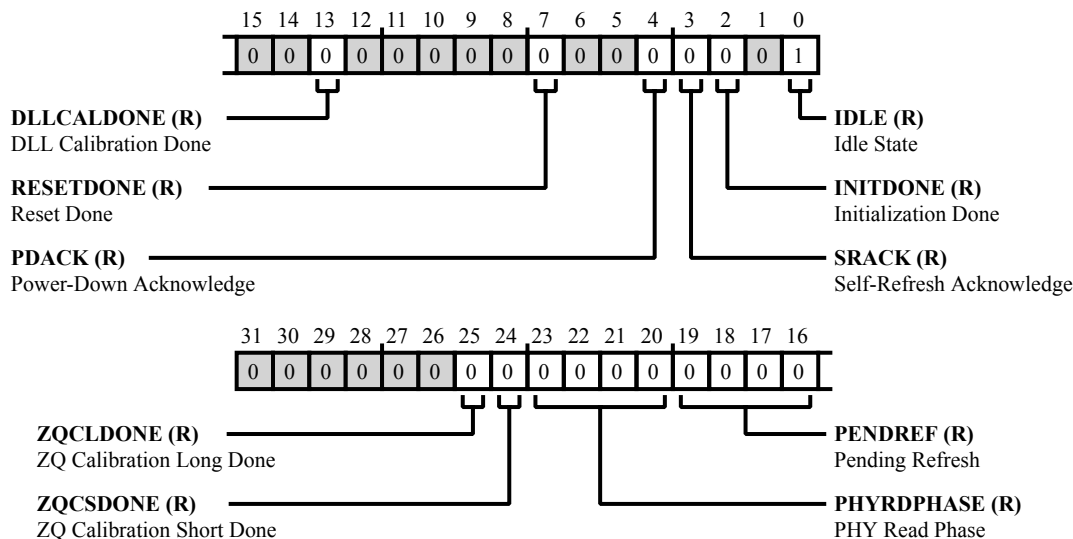


Figure 9-21: DMC_STAT Register Diagram

Table 9-26: DMC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	ZQCLDONE	ZQ Calibration Long Done. The <code>DMC_STAT.ZQCLDONE</code> bit checks if the ZQ calibration long sub routine is done.
		0 ZQ Calibration long is ongoing
		1 ZQ Calibration long is done
24 (R/NW)	ZQCSDONE	ZQ Calibration Short Done. The <code>DMC_STAT.ZQCSDONE</code> bit checks if the ZQ calibration short sub routine is done.
		0 ZQ Calibration short is ongoing
		1 ZQ Calibration Short is done
23:20 (R/NW)	PHYRDPHASE	PHY Read Phase. The <code>DMC_STAT.PHYRDPHASE</code> bits indicate the latency after which the DMC may read from the PHY. Taking round trip delay into account, the DLL indicates the exact number of clock cycles after which the controller needs to read data. Values other than those shown are reserved.
		2 2 Clock Cycles Latency

Table 9-26: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		3 3 Clock Cycles Latency
		4 4 Clock Cycles Latency
		5 5 Clock Cycles Latency
		6 6 Clock Cycles Latency
		7 7 Clock Cycles Latency
		8 8 Clock Cycles Latency
		9 9 Clock Cycles Latency
		10 10 Clock Cycles Latency
		11 11 Clock Cycles Latency
		12 12 Clock Cycles Latency
		13 13 Clock Cycles Latency
		14 14 Clock Cycles Latency
		15 15 Clock Cycles Latency
19:16 (R/NW)	PENDREF	<p>Pending Refresh.</p> <p>The DMC_STAT . PENDREF bits indicate the number of pending auto-refresh commands whose value can be from "0000" to "0111".</p>
13 (R/NW)	DLLCALDONE	<p>DLL Calibration Done.</p> <p>The DMC_STAT . DLLCALDONE indicates that the PHY DLL calibration sequence is complete.</p>
		0 No Status
		1 Completed PHY DLL Calibration
7 (R/NW)	RESETDONE	<p>Reset Done.</p> <p>The DMC_STAT . RESETDONE bit indicates that the reset sequence is complete.</p>
		0 SDRAM Reset is ongoing
		1 SDRAM Reset is done
4 (R/NW)	PDACK	<p>Power-Down Acknowledge.</p> <p>The DMC_STAT . PDACK bit indicates that power-down mode is active.</p>
		0 Not in Power-Down Mode
		1 Power-Down Mode Active

Table 9-26: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	SRACK	Self-Refresh Acknowledge. The DMC_STAT . SRACK bit indicates that self-refresh mode is active.
		0 Not in Self-Refresh Mode
		1 Self-Refresh Mode Active
2 (R/NW)	INITDONE	Initialization Done. The DMC_STAT . INITDONE bit indicates that the initialization sequence is complete.
		0 No Status
		1 Initialize Done
0 (R/NW)	IDLE	Idle State. The DMC_STAT . IDLE bit indicates whether the DMC is idle or busy.
		0 Busy
		1 Idle

Timing 0 Register

The `DMC_TR0` register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in `DMC_TR0` are in increments of clock cycle time (t_{CK}).

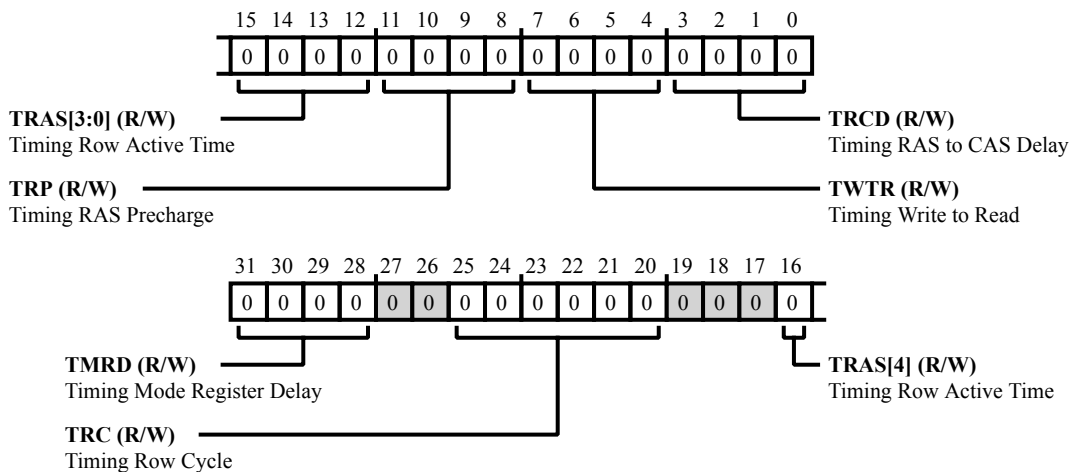


Figure 9-22: DMC_TR0 Register Diagram

Table 9-27: DMC_TR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/W)	TMRD	Timing Mode Register Delay. The <code>DMC_TR0.TMRD</code> field selects the set-to-active timing parameter (t_{MRD}), which is the number of clock cycles that occur after the mode registers in the SDRAM are set and before the next command is issued.
25:20 (R/W)	TRC	Timing Row Cycle. The <code>DMC_TR0.TRC</code> field selects the active-to-active time (t_{RC}), which is the minimum number of clock cycles that occur from an active command to the next active command in the same bank.
16:12 (R/W)	TRAS	Timing Row Active Time. The <code>DMC_TR0.TRAS</code> field selects the active-to-precharge time (t_{RAS}), which is the number of clock cycles that occur from an active command until a precharge command is allowed.
11:8 (R/W)	TRP	Timing RAS Precharge. The <code>DMC_TR0.TRP</code> field selects the precharge-to-active time (t_{RP}), which is the number of clock cycles that occur while the SDRAM recovers from a precharge command and becomes ready to accept the next active command.

Table 9-27: DMC_TR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	TWTR	Timing Write to Read. The DMC_TR0 . TWTR field selects the write-to-read delay time (t_{WTR}), which is the number of clock cycles that occur from the last write data to the next read command.
3:0 (R/W)	TRCD	Timing RAS to CAS Delay. The DMC_TR0 . TRCD field selects the RAS to CAS delay time (t_{RCD}), which is the number of clock cycles that occur from an active command to a read/write assertion.

Timing 1 Register

The `DMC_TR1` register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in `DMC_TR1` are in increments of clock cycle time (t_{CK}).

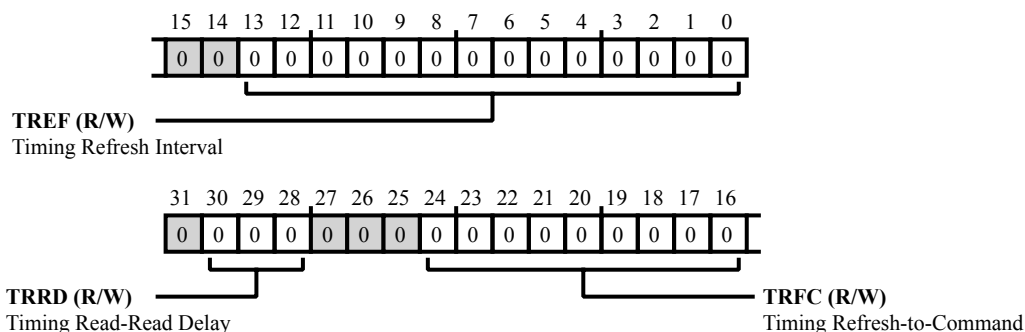


Figure 9-23: `DMC_TR1` Register Diagram

Table 9-28: `DMC_TR1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:28 (R/W)	TRRD	Timing Read-Read Delay. The <code>DMC_TR1</code> . <code>TRRD</code> field selects the active-to-active time (t_{RRD}), which is the minimum number of clock cycles occurring from a bank x active command to a bank y active command.
24:16 (R/W)	TRFC	Timing Refresh-to-Command. The <code>DMC_TR1</code> . <code>TRFC</code> field selects the refresh-to-active command delay (t_{RFC}), which is the number of clock cycles required for the SDRAM to recover from a refresh signal to be ready to take the next command. It is also the number of clock cycles needed for the SDRAM to recover from executing one active command and ready to accept the next active command.
13:0 (R/W)	TREF	Timing Refresh Interval. The <code>DMC_TR1</code> . <code>TREF</code> field selects the refresh interval time (t_{REF}), which is the number of clock cycles occurring from one refresh command to the next refresh command. The actual timing of issuing a precharge command may be delayed by if the SDRAM is processing a normal access. However, the delay is not accumulative so there is no need to shorten the refresh interval to account for the memory access time. The non-accumulative refresh delay typically increases memory bandwidth by a few percentage points.

Timing 2 Register

The `DMC_TR2` register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and before initializing the SDRAM.

Note that all values for bit fields in `DMC_TR2` are in increments of clock cycle time (t_{CK}).

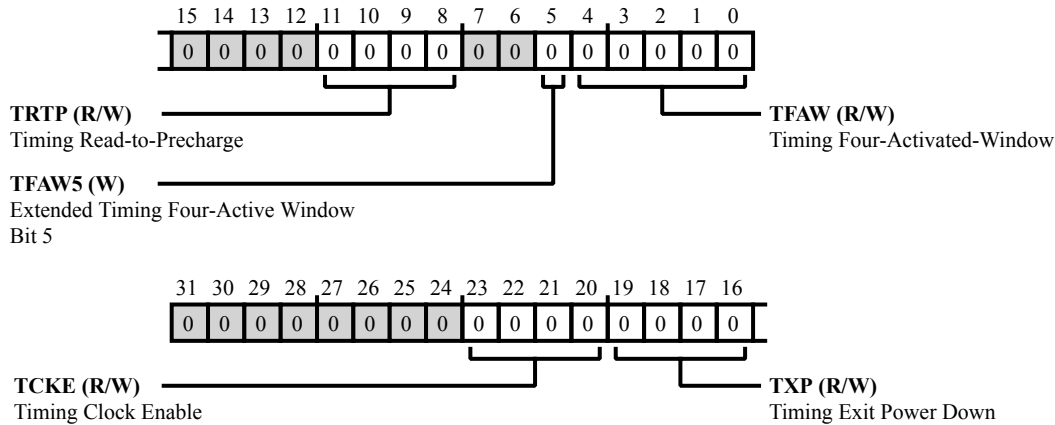


Figure 9-24: DMC_TR2 Register Diagram

Table 9-29: DMC_TR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:20 (R/W)	TCKE	Timing Clock Enable. The <code>DMC_TR2.TCKE</code> field selects the CKE minimum pulsewidth (t_{CKE}).
19:16 (R/W)	TXP	Timing Exit Power Down. The <code>DMC_TR2.TXP</code> field selects the exit power down to next valid command time (t_{XP}).
11:8 (R/W)	TRTP	Timing Read-to-Precharge. The <code>DMC_TR2.TRTP</code> field selects the internal read to precharge time (t_{RTP}). If the resulting value is less than 2, the register needs to be programmed with two. Note: Minimum t_{RTP} that needs to be programmed is 2.
5 (RX/W)	TFAW5	Extended Timing Four-Active Window Bit 5. The <code>DMC_TR2.TFAW5</code> bit is the extended bit for FAW timing for 800 MHz operation for values greater than 31. It is only applicable when <code>DMC_CTL.AL_EN = 1</code> . When <code>DMC_CTL.AL_EN = 0</code> , the <code>DMC_TR2.TFAW5</code> bit is reserved. When <code>DMC_CTL.AL_EN = 1</code> , the total $FAW[5:0] = \{TFAW5, TFAW[4:0]\}$. NOTE: This bit is write-only. Read as zero.

Table 9-29: DMC_TR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	TFAW	Timing Four-Activated-Window. The DMC_TR2 . TFAW field selects the four-banks-activated window time (t_{FAW}). No more than four SDRAM banks should be activated within this window.

ADSP-2156x DMC Register Descriptions

1600MHz DMCPHY (DDR3/DDR3L) (DMC) contains the following registers.

Table 9-30: ADSP-2156x DMC Register List

Name	Description
DMC_DDR_CA_CTL	DDR CA Lane Control Register
DMC_DDR_LANE0_CTL0	Data Lane 0 Control Register 0
DMC_DDR_LANE0_CTL1	Data Lane 0 Control Register 1
DMC_DDR_LANE1_CTL0	Data Lane 1 Control Register 0
DMC_DDR_LANE1_CTL1	Data Lane 1 Control Register 1
DMC_DDR_ROOT_CTL	DDR ROOT Module Control Register
DMC_DDR_ZQ_CTL0	DDR Calibration Control Register 0
DMC_DDR_ZQ_CTL1	DDR Calibration Control Register 1
DMC_DDR_ZQ_CTL2	DDR Calibration Control Register 2

DDR CA Lane Control Register

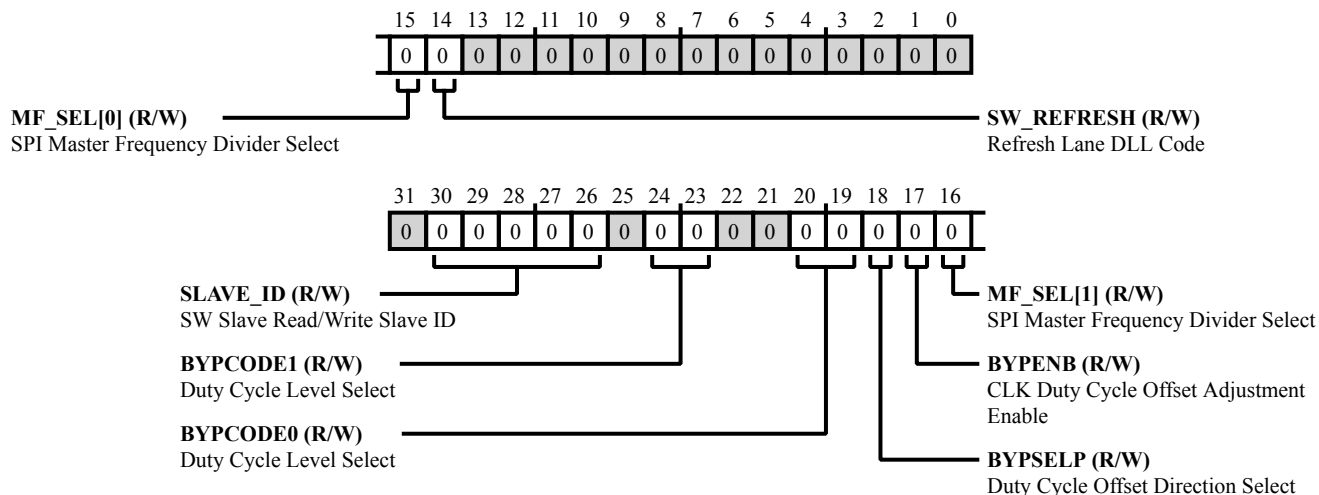


Figure 9-25: DMC_DDR_CA_CTL Register Diagram

Table 9-31: DMC_DDR_CA_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:26 (R/W)	SLAVE_ID	SW Slave Read/Write Slave ID. The DMC_DDR_CA_CTL.SLAVE_ID bit field is the slave ID for SW slave read or write transactions.
24:23 (R/W)	BYPCODE1	Duty Cycle Level Select. The DMC_DDR_CA_CTL.BYPCODE1 bit field combines with BYPCODE0 to select one of four levels of duty cycle adjustment.
		0
		0
		1
20:19 (R/W)	BYPCODE0	Duty Cycle Level Select. The DMC_DDR_CA_CTL.BYPCODE0 bit field combines with BYPCODE1 to select one of four levels of duty cycle adjustment.
		0
		0
		1
		2

Table 9-31: DMC_DDR_CA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	BYPSELP	Duty Cycle Offset Direction Select. The DMC_DDR_CA_CTL.BYPSELP bit selects the direction of the duty cycle offset.
		0 Negative
		1 Positive
17 (R/W)	BYPENB	CLK Duty Cycle Offset Adjustment Enable. The DMC_DDR_CA_CTL.BYPENB bit enables the CLK duty cycle offset adjustment.
		0 Disable
		1 Enable
16:15 (R/W)	MF_SEL	SPI Master Frequency Divider Select. The DMC_DDR_CA_CTL.MF_SEL bit selects the SPI master frequency divider for root to lane transfers.
		0 Divide by 2
		1 Divide by 4
		2 Divide by 8
		3 Divide by 16
14 (R/W)	SW_REFRESH	Refresh Lane DLL Code.

Data Lane 0 Control Register 0

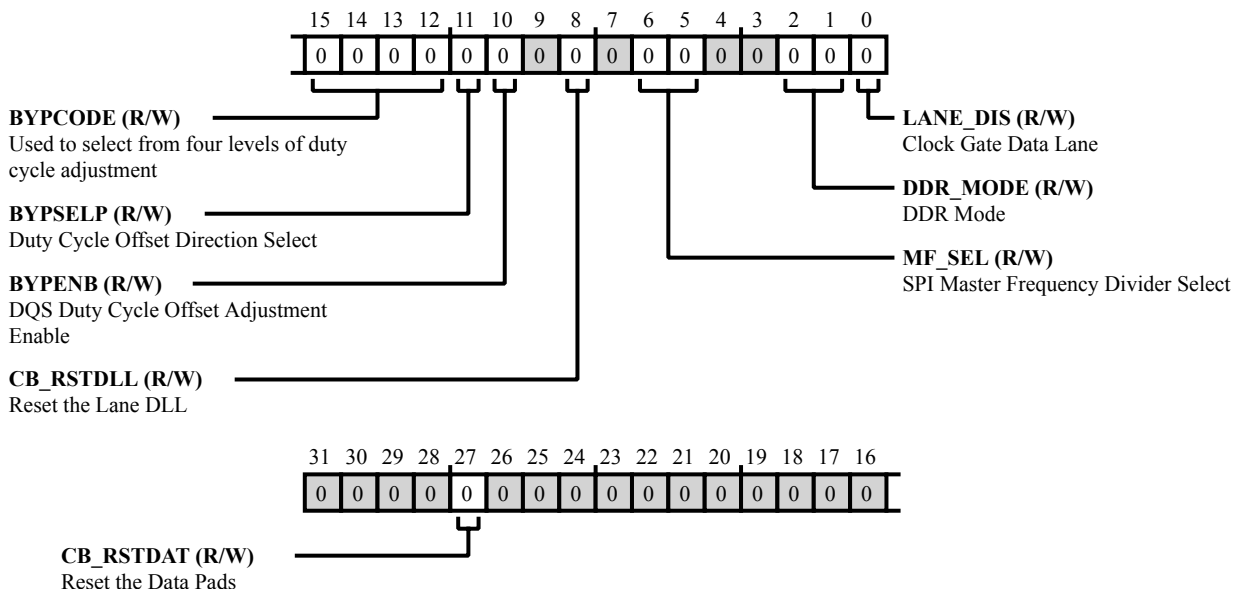


Figure 9-26: DMC_DDR_LANE0_CTL0 Register Diagram

Table 9-32: DMC_DDR_LANE0_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	CB_RSTDAT	Reset the Data Pads.	
15:12 (R/W)	BYPCODE	Used to select from four levels of duty cycle adjustment.	
		1	
		2	
		4	
11 (R/W)	BYPSEL	Duty Cycle Offset Direction Select. The DMC_DDR_LANE0_CTL0.BYPSEL bit selects the direction of the duty cycle offset.	
		0	Negative
		1	Positive

Table 9-32: DMC_DDR_LANE0_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	BYPENB	DQS Duty Cycle Offset Adjustment Enable. The DMC_DDR_LANE0_CTL0 .BYPENB bit enables the DQS duty cycle offset adjustment.
		0 Disable
		1 Enable
8 (R/W)	CB_RSTDLL	Reset the Lane DLL.
6:5 (R/W)	MF_SEL	SPI Master Frequency Divider Select. The DMC_DDR_LANE0_CTL0 .MF_SEL bit selects the SPI master frequency divider for root to lane transfers.
		0 Divide by 2
		1 Divide by 4
		2 Divide by 8
		3 Divide by 16
2:1 (R/W)	DDR_MODE	DDR Mode.
		0 DDR3/3L Mode
0 (R/W)	LANE_DIS	Clock Gate Data Lane.
		0 Lane is Enabled
		1 Lane is Clock Gated

Data Lane 0 Control Register 1

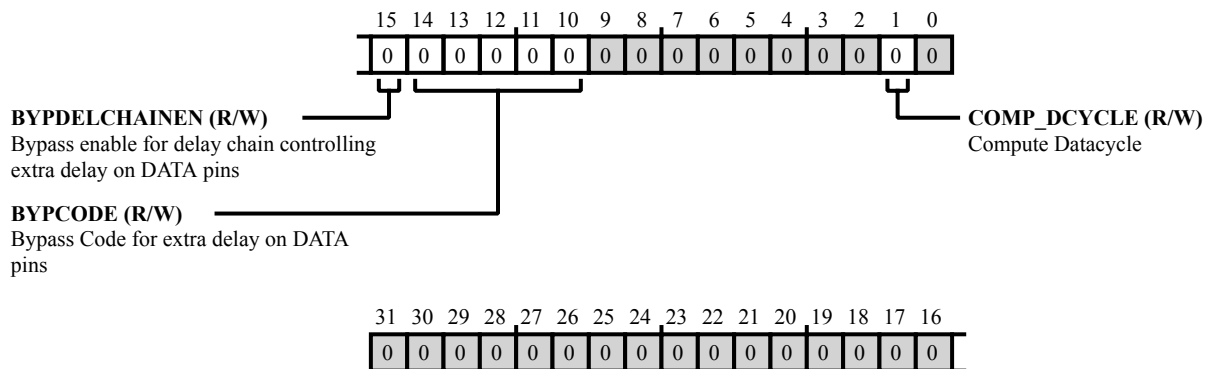


Figure 9-27: DMC_DDR_LANE0_CTL1 Register Diagram

Table 9-33: DMC_DDR_LANE0_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	BYPDELCHAINEN	Bypass enable for delay chain controlling extra delay on DATA pins.
14:10 (R/W)	BYPCODE	Bypass Code for extra delay on DATA pins.
1 (R/W)	COMP_DCYCLE	Compute Datacycle.

Data Lane 1 Control Register 0

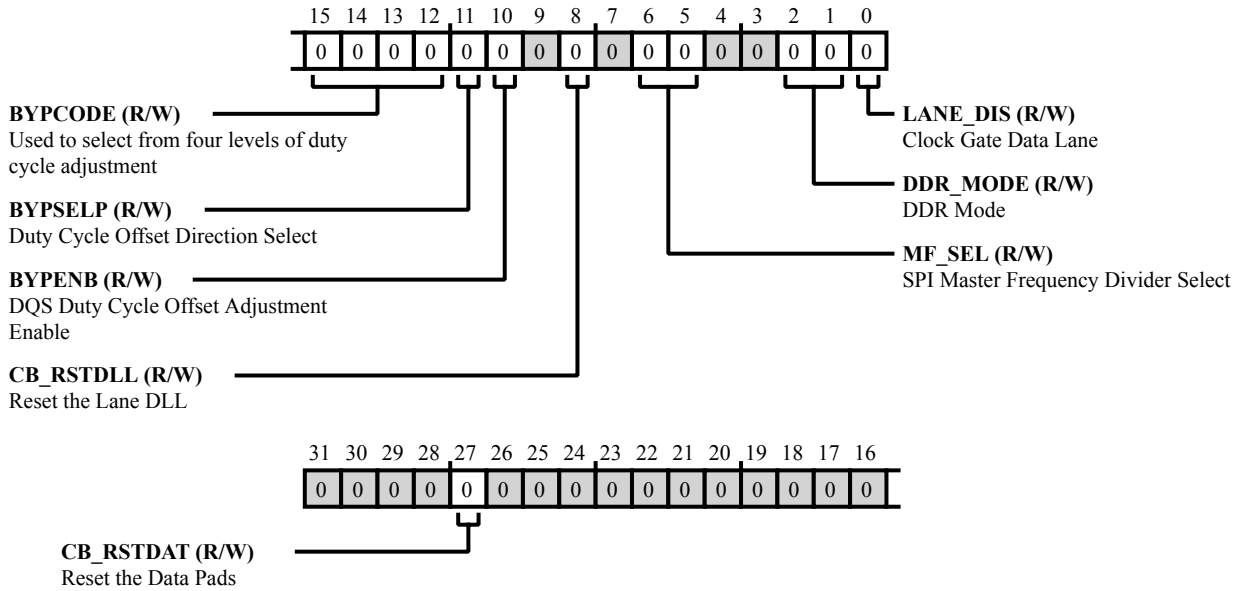


Figure 9-28: DMC_DDR_LANE1_CTL0 Register Diagram

Table 9-34: DMC_DDR_LANE1_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	CB_RSTDAT	Reset the Data Pads.	
15:12 (R/W)	BYPCODE	Used to select from four levels of duty cycle adjustment.	
		1	
		2	
		4	
11 (R/W)	BYPSEL	Duty Cycle Offset Direction Select. The DMC_DDR_LANE1_CTL0.BYPSEL bit selects the direction of the duty cycle offset.	
		0	Negative
		1	Positive

Table 9-34: DMC_DDR_LANE1_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	BYPENB	DQS Duty Cycle Offset Adjustment Enable. The DMC_DDR_LANE1_CTL0 .BYPENB bit enables the DQS duty cycle offset adjustment.
		0 Disable
		1 Enable
8 (R/W)	CB_RSTDLL	Reset the Lane DLL.
6:5 (R/W)	MF_SEL	SPI Master Frequency Divider Select. The DMC_DDR_LANE1_CTL0 .MF_SEL bit selects the SPI master frequency divider for root to lane transfers.
		0 Divide by 2
		1 Divide by 4
		2 Divide by 8
		3 Divide by 16
2:1 (R/W)	DDR_MODE	DDR Mode.
		0 DDR3/3L Mode
0 (R/W)	LANE_DIS	Clock Gate Data Lane.
		0 Lane is Enabled
		1 Lane is Clock Gated

Data Lane 1 Control Register 1

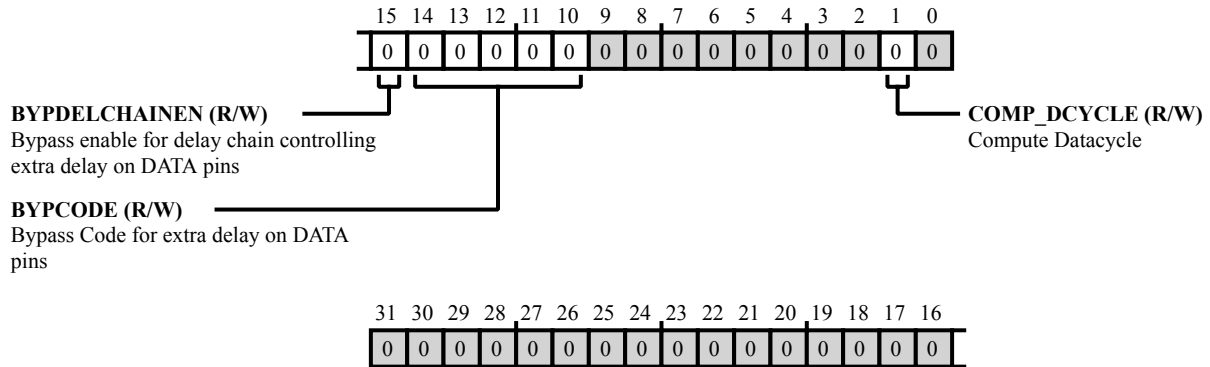


Figure 9-29: DMC_DDR_LANE1_CTL1 Register Diagram

Table 9-35: DMC_DDR_LANE1_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	BYPDELCHAINEN	Bypass enable for delay chain controlling extra delay on DATA pins.
14:10 (R/W)	BYPCODE	Bypass Code for extra delay on DATA pins.
1 (R/W)	COMP_DCYCLE	Compute Datacycle.

DDR ROOT Module Control Register

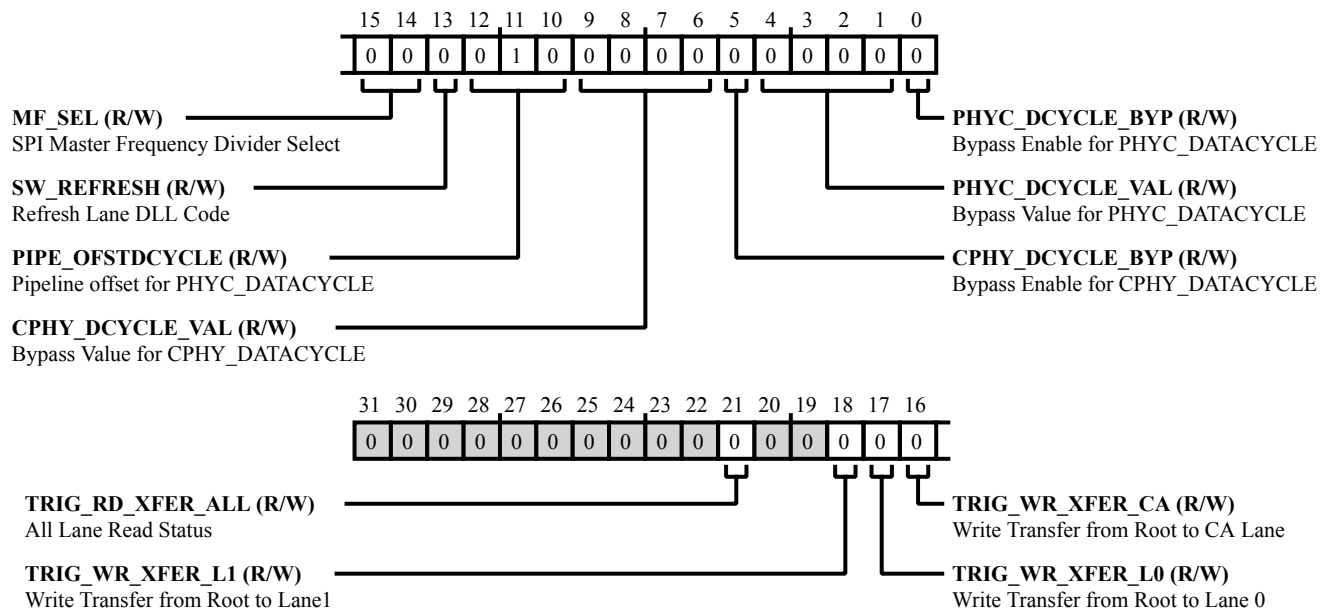


Figure 9-30: DMC_DDR_ROOT_CTL Register Diagram

Table 9-36: DMC_DDR_ROOT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	TRIG_RD_XFER_ALL	All Lane Read Status. Software trigger to read the status data from all lanes.
18 (R/W)	TRIG_WR_XFER_L1	Write Transfer from Root to Lane1. Software trigger for a write transfer from the root to lane 1.
17 (R/W)	TRIG_WR_XFER_L0	Write Transfer from Root to Lane 0. Software trigger for a write transfer from the root to lane 0.
16 (R/W)	TRIG_WR_XFER_CA	Write Transfer from Root to CA Lane. Software trigger for a write transfer from the root to CA lane.
15:14 (R/W)	MF_SEL	SPI Master Frequency Divider Select. The DMC_DDR_ROOT_CTL.MF_SEL bit selects the SPI master frequency divider for root to lane transfers.
		0 Divide by 2
		1 Divide by 4
		2 Divide by 8
		3 Divide by 16

Table 9-36: DMC_DDR_ROOT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	SW_REFRESH	Refresh Lane DLL Code.
12:10 (R/W)	PIPE_OFSTDCYCLE	Pipeline offset for PHYC_DATACYCLE.
9:6 (R/W)	CPHY_DCYCLE_VAL	Bypass Value for CPHY_DATACYCLE.
5 (R/W)	CPHY_DCYCLE_BYP	Bypass Enable for CPHY_DATACYCLE.
4:1 (R/W)	PHYC_DCYCLE_VAL	Bypass Value for PHYC_DATACYCLE.
0 (R/W)	PHYC_DCYCLE_BYP	Bypass Enable for PHYC_DATACYCLE.

DDR Calibration Control Register 0

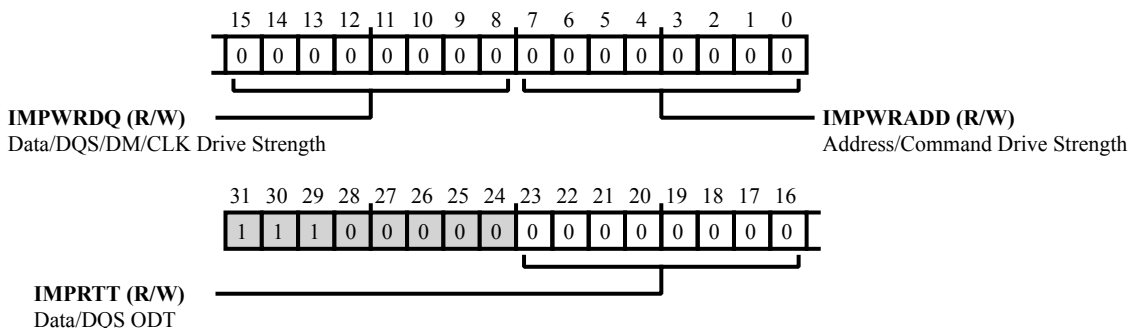


Figure 9-31: DMC_DDR_ZQ_CTL0 Register Diagram

Table 9-37: DMC_DDR_ZQ_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	IMPRTT	Data/DQS ODT. Desired ODT in ohms for Data and DQS pads.
15:8 (R/W)	IMPWRDQ	Data/DQS/DM/CLK Drive Strength. Desired drive strength in ohms for Data, DQS, DM, and clock pads. A drive strength of 100 ohms is recommended.
7:0 (R/W)	IMPWRADD	Address/Command Drive Strength. Desired drive strength in ohms for address and command pads. A drive strength of 100 ohms is recommended.

DDR Calibration Control Register 1

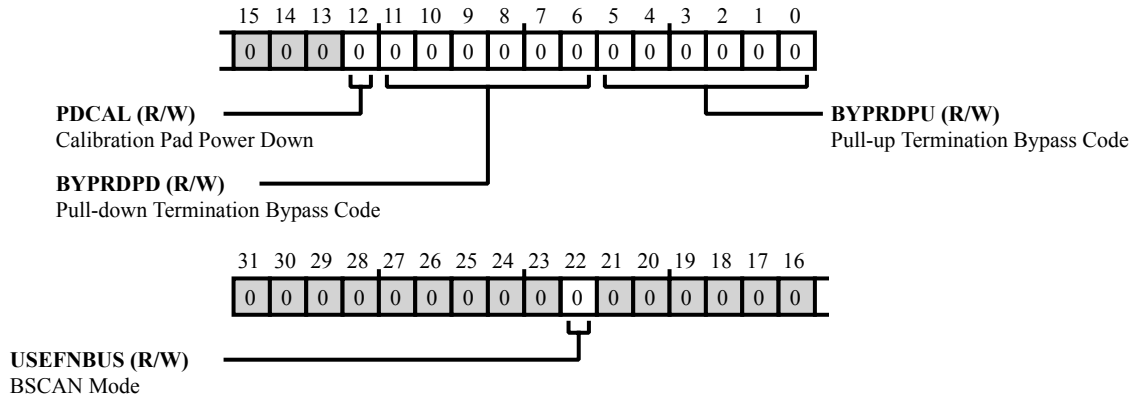


Figure 9-32: DMC_DDR_ZQ_CTL1 Register Diagram

Table 9-38: DMC_DDR_ZQ_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	USEFNBUS	BSCAN Mode. The bit controls whether to use bypass codes or functional codes in BSCAN mode.
12 (R/W)	PDCAL	Calibration Pad Power Down. Software bypass to power down the calibration pad.
11:6 (R/W)	BYPRDPD	Pull-down Termination Bypass Code.
5:0 (R/W)	BYPRDPU	Pull-up Termination Bypass Code.

DDR Calibration Control Register 2

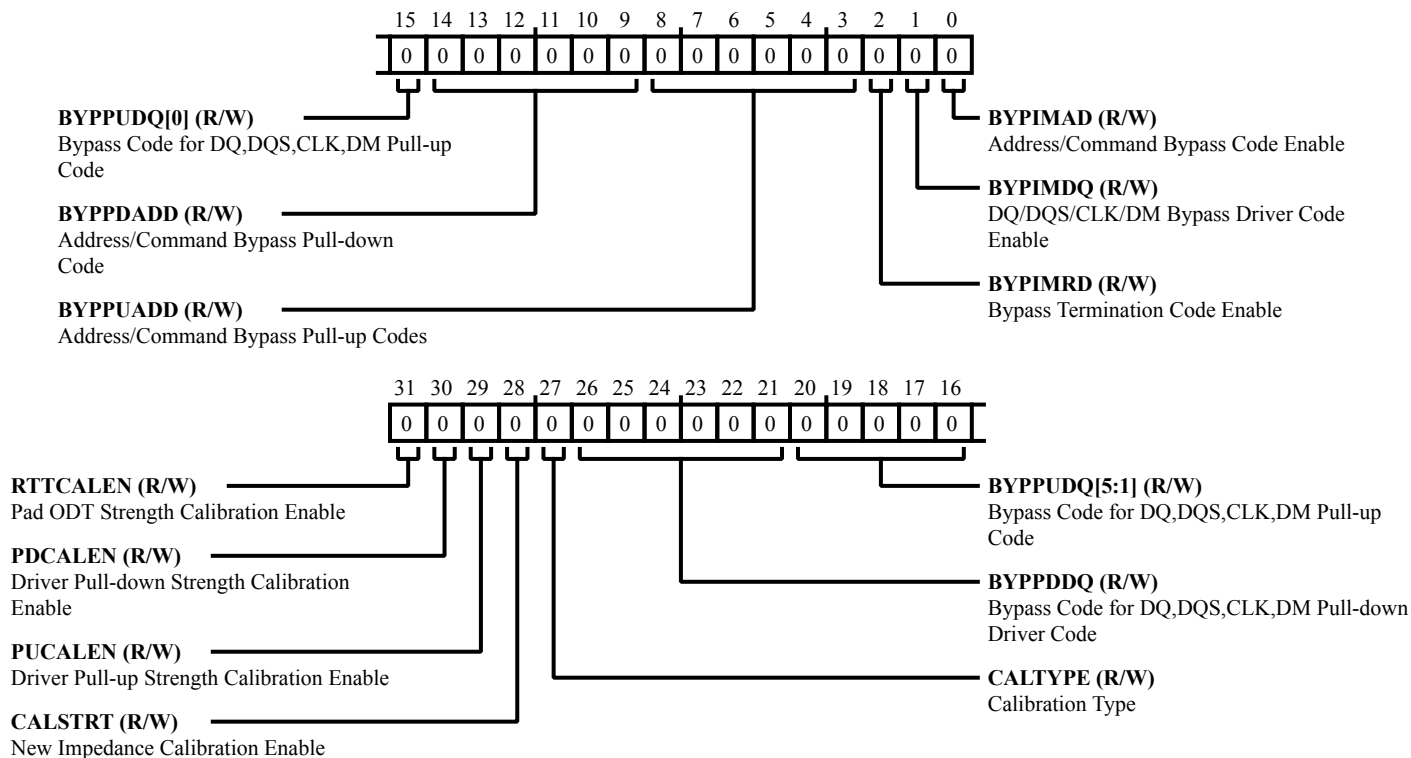


Figure 9-33: DMC_DDR_ZQ_CTL2 Register Diagram

Table 9-39: DMC_DDR_ZQ_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	RTTCALEN	Pad ODT Strength Calibration Enable.
30 (R/W)	PDCALEN	Driver Pull-down Strength Calibration Enable.
29 (R/W)	PUCALEN	Driver Pull-up Strength Calibration Enable.
28 (R/W)	CALSTRT	New Impedance Calibration Enable.
27 (R/W)	CALTYPE	Calibration Type. Short or Long Calibration
26:21 (R/W)	BYPPDDQ	Bypass Code for DQ,DQS,CLK,DM Pull-down Driver Code.

Table 9-39: DMC_DDR_ZQ_CTL2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20:15 (R/W)	BYPPUDQ	Bypass Code for DQ,DQS,CLK,DM Pull-up Code.
14:9 (R/W)	BYPPDADD	Address/Command Bypass Pull-down Code.
8:3 (R/W)	BYPPUADD	Address/Command Bypass Pull-up Codes. Bypass pull-up codes for address and command pads
2 (R/W)	BYPIMRD	Bypass Termination Code Enable.
1 (R/W)	BYPIMDQ	DQ/DQS/CLK/DM Bypass Driver Code Enable.
0 (R/W)	BYPIPAD	Address/Command Bypass Code Enable. Bypass code enable for address and command pads.

10 One-Time Programmable Memory Controller (OTPC)

This chapter describes the operation of the OTP controller. The OTP module is a complete system integrating an OTP memory core with a programming controller, charge pump, and voltage regulator. A built-in Hamming Code Error Correction (ECC), and a fully implemented double-redundant program or read scheme protect the OTP data.

OTP memory access is through the [OTP API Overview](#) provided by the ROM.

CAUTION: OTP memory does not support burst transfers, which are required to support cache line fills. As such, OTP memory should not be made cacheable. If it is, the OTP controller returns an error when a read access is attempted.

OTPC Features

The OTP memory and controller have the following features:

- Built-in redundant read mode
- Built-in integrated power supply
- Built-in Hamming Code Error Correction (ECC)
- Full word serial (single bit at a time) programming with internal VPP

OTPC Functional Description

ADSP-2156x OTPC Register List

The One-Time-Programmable Memory controller (OTPC) supports programming the OTP memory. A set of registers governs OTPC operations. For more information on OTPC functionality, see the OTPC register descriptions.

Table 10-1: ADSP-2156x OTPC Register List

Name	Description
OTPC_SECU_STATE	OTP Security State Register

Table 10-1: ADSP-2156x OTPC Register List (Continued)

Name	Description
OTPC_STAT	OTP Status Register

ADSP-2156x OTPC Interrupt List

Table 10-2: ADSP-2156x OTPC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
5	OTPC0_ERR	OTPC0 Dual-bit error	Level	

Error Correction

The OTP memory features a Hamming error correction implementation. Signal bit errors are automatically corrected, and dual-bit errors are detected. Refer to [OTPC Interrupt Signals](#).

ECC is always enabled. ECC applies to each 16-bit segment. Because of this functionality, each 16-bit location can only be written to once. Writing to a 16-bit location a second time results in unexpected behavior.

OTP Layout

This section details the memory layout of the OTP memory.

Table 10-3: ADSP-2156x OTP Layout

Name	32-Bit Aligned Byte Address	Size (bits)	Description
reserved	0x0 + 0 – 0x1C	256	reserved
ospi_read_data_capture	0x74 + 0 – 0x74	16	OSPI Read Data Capture register
reserved	0x74 + 16 – 0x74	16	reserved
huk	0x78 + 0 – 0x94	256	Hardware Unique Key
reserved	0xB0 + 0 – 0x14C	1280	reserved
pvt_128key0	0x168 + 0 – 0x174	128	Customer Privatekey 0 128bits
pvt_128key1	0x178 + 0 – 0x184	128	Customer Privatekey 1 128bits
pvt_128key2	0x188 + 0 – 0x194	128	Customer Privatekey 2 128bits
pvt_128key3	0x198 + 0 – 0x1A4	128	Customer Privatekey 3 128bits
ek	0x1A8 + 0 – 0x1C4	256	Endorsement Key
secure_emu_key	0x1C8 + 0 – 0x1D4	128	Secure Emulation Key
reserved	0x1DC + 0 – 0x1EC	160	reserved

Table 10-3: ADSP-2156x OTP Layout (Continued)

Name	32-Bit Aligned Byte Address	Size (bits)	Description
reserved	0x1F8 + 0 – 0x1FC	48	reserved
emu_key_disable	0x1FC + 16 – 0x1FC	16	Secure Emulation Key Disable
public_key0	0x200 + 0 – 0x23C	512	Customer Public Key0
public_key1	0x240 + 0 – 0x27C	512	Customer Public Key1
cgu_ctl_WEN	0x280 + 0 – 0x280	1	CGU Config ctl_WEN
cgu_div_WEN	0x280 + 1 – 0x280	1	CGU Config div_WEN
cgu_reserved0	0x280 + 2 – 0x280	1	CGU Config reserved0
cgu_div_DSEL	0x280 + 3 – 0x280	5	CGU Config div_DSEL
cgu_div_CSEL	0x280 + 8 – 0x280	5	CGU Config div_CSEL
cgu_div_S0SEL	0x280 + 13 – 0x280	3	CGU Config div_S0SEL
cgu_div_SYSSSEL	0x280 + 16 – 0x280	5	CGU Config div_SYSSSEL
cgu_div_S1SEL	0x280 + 21 – 0x280	3	CGU Config div_S1SEL
cgu_div_OSEL	0x280 + 24 – 0x280	7	CGU Config div_OSEL
cgu_ctl_DF	0x280 + 31 – 0x280	1	CGU Config ctl_DF
cgu_ctl_MSEL	0x284 + 0 – 0x284	7	CGU Config ctl_MSEL
cgu_auto_disable	0x284 + 7 – 0x284	1	CGU Disable Auto Alignment
cgu_reserved1	0x284 + 8 – 0x284	6	CGU Config reserved1
cgu_clkoutssel_CLKOUTSEL	0x284 + 14 – 0x284	5	CGU Config clkoutssel_CLKOUTSEL
cgu_clkoutssel_WEN	0x284 + 19 – 0x284	1	CGU Config clkoutssel_WEN
cgu_reserved2	0x284 + 20 – 0x284	12	CGU Config reserved2
cgu_oscwctl0_WEN	0x288 + 0 – 0x288	1	CGU Config oscwctl0_WEN
cgu_oscwctl0_HODF	0x288 + 1 – 0x288	6	CGU Config oscwctl0_HODF
cgu_oscwctl0_HODEN	0x288 + 7 – 0x288	1	CGU Config oscwctl0_HODEN
cgu_oscwctl0_CNGEN	0x288 + 8 – 0x288	1	CGU Config oscwctl0_CNGEN
cgu_oscwctl0_BOUF	0x288 + 9 – 0x288	5	CGU Config oscwctl0_BOUF
cgu_oscwctl0_BOUEN	0x288 + 14 – 0x288	1	CGU Config oscwctl0_BOUEN
cgu_oscwctl0_FAULTEN	0x288 + 15 – 0x288	1	CGU Config oscwctl0_FAULTEN
cgu_oscwctl0_MONDIS	0x288 + 16 – 0x288	1	CGU Config oscwctl0_MONDIS
cgu_oscwctl0_FAULTPINDIS	0x288 + 17 – 0x288	1	CGU Config oscwctl0_FAULTPINDIS
cgu_reserved3	0x288 + 18 – 0x288	14	CGU Config reserved3
flashStartAddress	0x28C + 0 – 0x28C	32	Flash Start Address Override

Table 10-3: ADSP-2156x OTP Layout (Continued)

Name	32-Bit Aligned Byte Address	Size (bits)	Description
spiMasterBootCmd	0x290 + 0 – 0x290	32	SPI Master Boot Command Override
spiSlaveBootCmd	0x294 + 0 – 0x294	32	SPI Slave Boot Command Override
lpBootCmd	0x298 + 0 – 0x298	32	Link Port Boot Command Override
uartBootCmd	0x29C + 0 – 0x29C	32	UART Boot Command Override
ospiMasterBootCmd	0x2A0 + 0 – 0x2A0	32	OSPI Master Boot Command Override
bcfg_lock_monitor	0x2A4 + 0 – 0x2A4	1	Boot Config lock_monitor
bcfg_reserved2	0x2A4 + 1 – 0x2A4	15	Boot Config reserved2
bcfg_reserved3	0x2A4 + 16 – 0x2A4	16	Boot Config reserved3
bcfg_pubkey0Inv	0x2A8 + 0 – 0x2A8	1	Boot Config pubkey0Inv
bcfg_reserved4	0x2A8 + 1 – 0x2A8	15	Boot Config reserved4
bcfg_pubkey1Inv	0x2A8 + 16 – 0x2A8	1	Boot Config pubkey1Inv
bcfg_reserved5	0x2A8 + 17 – 0x2A8	15	Boot Config reserved5
bcfg_privkey0Inv	0x2AC + 0 – 0x2AC	1	Boot Config privkey0Inv
bcfg_reserved6	0x2AC + 1 – 0x2AC	15	Boot Config reserved6
bcfg_privkey1Inv	0x2AC + 16 – 0x2AC	1	Boot Config privkey1Inv
bcfg_reserved7	0x2AC + 17 – 0x2AC	15	Boot Config reserved7
bcfg_privkey2Inv	0x2B0 + 0 – 0x2B0	1	Boot Config privkey2Inv
bcfg_reserved8	0x2B0 + 1 – 0x2B0	15	Boot Config reserved8
bcfg_privkey3Inv	0x2B0 + 16 – 0x2B0	1	Boot Config privkey3Inv
bcfg_reserved9	0x2B0 + 17 – 0x2B0	15	Boot Config reserved9
bcfg_dmcEn	0x2B4 + 0 – 0x2B4	1	Boot Config dmcEn
bcfg_reserved10	0x2B4 + 1 – 0x2B4	15	Boot Config reserved10
bcfg_dmcInv	0x2B4 + 16 – 0x2B4	1	Boot Config dmcInv
bcfg_reserved11	0x2B4 + 17 – 0x2B4	15	Boot Config reserved11
reserved	0x2B8 + 0 – 0x2BC	64	reserved
antiroll_nv_cntr	0x2C0 + 0 – 0x2FC	512	AntiRollback NV Counter
gp1	0x300 + 0 – 0x33C	512	General Purpose 1
reserved	0x340 + 0 – 0x340	24	reserved
bootModeDisable	0x340 + 24 – 0x340	8	Boot Mode Disable Bits
preboot_ddr_cfg	0x344 + 0 – 0x370	384	User PrebootDDR configuration
stageID	0x374 + 0 – 0x378	48	StageID

Table 10-3: ADSP-2156x OTP Layout (Continued)

Name	32-Bit Aligned Byte Address	Size (bits)	Description
reserved	0x378 + 16 – 0x378	16	reserved
lock	0x48C + 0 – 0x48C	1	Lockbit
Reserved	0x48C + 1 – 0x48C	31	Reserved
Reserved	0x490 + 0 – 0x4AD	928	Reserved

OTPC Event Control

The following sections provide information on OTP events and error management.

OTPC Interrupt Signals

When making 32-bit accesses to OTP memory, a double-bit error in any 16-bit segment triggers the OTPC_INT interrupt. The OTPC also has the OTPC dual bit error (OTPC0_ERR) with the SEC ID of 5. See the [System Event Controller \(SEC\)](#) chapter for more information.

OTPC Status and Error Signals

The OTP controller does not produce error signals.

OTP API Overview

The ROM provides a set of functions to facilitate OTP field access. The OTP memory is broken up into a set of specialized fields that are described in this section. The API removes the requirement of understanding the details of the layout or OTP access procedures.

All OTP accesses are made through the provided API.

OTP Programming

The OTP programming API provides a simple access, abstracting particulars of the OTP controller.

Any fields that contain zero or null pointers are skipped.

All addresses are assumed to be byte addresses unless otherwise noted.

A list of APIs follows:

<code>bool adi_rom_otp_pgm(otp_data* data);</code>	OTP Program
<code>bool adi_rom_lock();</code>	Lock API

OTP Program

Program OTP memory using a struct containing the following predefined data fields.

Name	OTP Program	-
PP Define	FUNC_ROM_OTPPGM	
Prototype	bool adi_rom_otp_pgm(otp_data* data);	-
Argument	data	struct containing data to program OTP with
Return Value	bool	true for programming success
Stack Requirements	valid stack	-

```
bool res = adi_rom_otp_pgm(data);
```

The following type of struct is available for programming. Refer to the ROM header file for the exact definition

```
typedef struct {
    void *reserved2;
    uint32_t ospi_read_data_capture:16;
    uint32_t (*huk)[ROM_OTP_SZ_huk];
    uint32_t (*pvt_128key0)[ROM_OTP_SZ_pvt_128key0];
    uint32_t (*pvt_128key1)[ROM_OTP_SZ_pvt_128key1];
    uint32_t (*pvt_128key2)[ROM_OTP_SZ_pvt_128key2];
    uint32_t (*pvt_128key3)[ROM_OTP_SZ_pvt_128key3];
    uint32_t (*ek)[ROM_OTP_SZ_ek];
    uint32_t (*secure_emu_key)[ROM_OTP_SZ_secure_emu_key];
    uint32_t emu_key_disable:16;
    uint32_t (*public_key0)[ROM_OTP_SZ_public_key0];
    uint32_t (*public_key1)[ROM_OTP_SZ_public_key1];
    uint32_t (*boot_info)[ROM_OTP_SZ_boot_info];
    uint8_t antiroll_nv_cntr;
    uint32_t (*gp1)[ROM_OTP_SZ_gp1];
    uint32_t bootModeDisable:8;
    uint32_t (*preboot_ddr_cfg)[ROM_OTP_SZ_preboot_ddr_cfg];
    uint32_t (*stageID)[ROM_OTP_SZ_stageID];
} otp_data;
```

NOTE: Make OTP memory a non-cacheable region if the core needs access to it.

OTP Reading

This API provides a unified source for retrieving OTP data fields.

All addresses are assumed to be byte addresses, unless otherwise noted.

A list of APIs follow:

bool adi_rom_otp_get(OTPCMD cmd, uint32_t* data);	OTP Get Field
---	-------------------------------

OTP Get Field

Retrieves indicated data from OTP memory.

Name	OTP Get Field	
Prototype	<code>bool adi_rom_otp_get(OTPCMD cmd, uint32_t* data);</code>	
Argument	<code>cmd</code>	Indicates what data to fetch, based on the <code>OTPCMD</code> enum.
Argument	<code>data</code>	memory location to write the data to
Return Value	<code>bool</code>	true for a successful read
Stack Requirements	valid stack	

```
bool res = adi_rom_otp_get(otpcmd_info,data);
```

The data specified by the `OTPCMD` enum parameter is fetched from OTP memory and placed in the location specified by `data`. The `OTPCMD` enum contains entries for each field defined in OTP memory, for the most current list, refer to the OTP header file.

An example of the enum style follows:

```
typedef enum {
    otpcmd_reserved0 = 0,
    otpcmd_reserved2,
    otpcmd_ospi_read_data_capture, /*!< OSPI Read Data Capture Register */
    otpcmd_huk, /*!< Hardware Unique Key */
    otpcmd_pvt_128key0, /*!< Customer Privatekey 0 128bits */
    otpcmd_pvt_128key1, /*!< Customer Privatekey 1 128bits */
    otpcmd_pvt_128key2, /*!< Customer Privatekey 2 128bits */
    otpcmd_pvt_128key3, /*!< Customer Privatekey 3 128bits */
    otpcmd_ek, /*!< Endorsement Key */
    otpcmd_secure_emu_key, /*!< Secure Emulation Key */
}
```

OTP Counters

The OTPC module implements a counter API to allow easy reading or writing of the counter without dealing with the complexities of rewriting OTP memory sections that are ECC protected.

The OTPC module provides two functional APIs for counters. These APIs are not extra; the module uses the same `get` and `pgm` APIs. The APIs are functionally unique in the way that they set and retrieve data as counters in OTP memory.

The API uses a different method to count bits because each bit in OTP memory can only be set =1 once, and the ECC protects each 16-bit unit. This functionality essentially means that each 16-bit unit can only be written to once. Therefore, a counter that can count 0–31 requires 32×16 bits of memory.

The API receives and returns the value of the counter as a `uint8_t` binary number. Writing a value less than the current value of the counter or greater than the maximum value results in an error.

To implement this functionality, the driver counts by shifting 1's from the left, treating each block as 1 bit. A three-bit counter is encoded as follows.

bit 2	bit 1	bit 0	Value
0000	0000	0000	0
0001	0000	0000	1
0011	0001	0000	2
0011	0001	0001	3

Lock API

This API locks the device.

Name	Lock API	-
PP Define	<code>FUNC_ROM_LOCK</code>	-
Prototype	<code>bool adi_rom_lock();</code>	-
Return Value	<code>bool</code>	true for success
Stack Requirements	valid stack	-

```
bool res = adi_rom_lock();
```

Calling this function locks the device, making it a secure. Once locked, the [OTPC_SECU_STATE](#) register indicates that the part is locked, and access is limited. For more information, refer to the security documentation regarding a locked device.

NOTE: Locked Status. The [OTPC_SECU_STATE](#) register is updated only after the part is rebooted. After calling the lock function, the register still indicates that the part is open.

ADSP-2156x OTPC Register Descriptions

OTP Memory Controller (OTPC) contains the following registers.

Table 10-4: ADSP-2156x OTPC Register List

Name	Description
OTPC_SECU_STATE	OTP Security State Register
OTPC_STAT	OTP Status Register

OTPC Security State Register

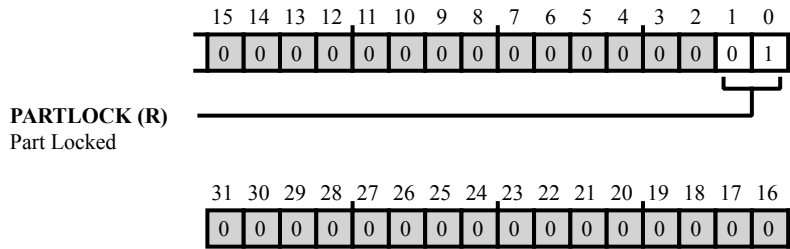


Figure 10-1: OTPC_SECU_STATE Register Diagram

Table 10-5: OTPC_SECU_STATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/NW)	PARTLOCK	Part Locked. The <code>OTPC_SECU_STATE.PARTLOCK</code> bit indicates a locked part.
		0 OPEN part
		1 Locked part
		2 Reserved

OTPC Status Register

The `OTPC_STAT` register bits indicate errors and flag status and control the protection bits.

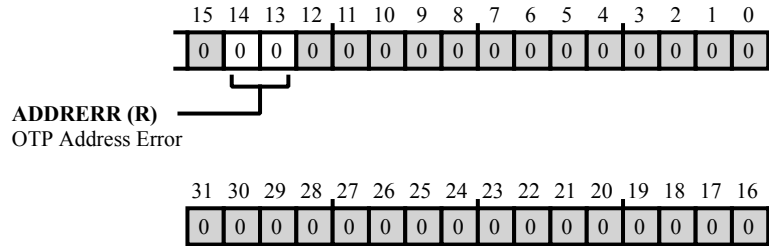


Figure 10-2: OTPC_STAT Register Diagram

Table 10-6: OTPC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:13 (R/NW)	ADDRERR	OTPC Address Error. The <code>OTPC_STAT.ADDRERR</code> bit field indicates errors which occur when the OTP programming address is out of range or tries to access protected space.
		0 No error - proper OTP address
		1 OTP address out of range
		2 8-bit OTP address
		3 Protected OTP address

11 System Memory Protection Unit (SMPU)

The SMPU provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

SMPU Features

The system memory protection unit has the following features.

- After reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces by any master.
- Each SMPU instance can be configured to monitor multiple regions. Each can be individually enabled.
- Each region can be configured with its own protection settings.
- Provides general read or write protection.
- Read and write transactions are restricted or allowed depending on the transaction ID.

On the ADSP-2156x processor, four SMPU instances are available to protect the L2, external memory (DMC) interface.

Table 11-1: SMPU Instances

Module	SMPU Instance
Core_L2	2
DMA_L2	3
DMC0	9
SPI Flash Address Space	11

All the SMPU instances can be configured up to eight regions.

- Up to eight outstanding read or write transactions supported on SMPU instances for Core_L2 and DMC0. Up to four outstanding read or write transactions supported on the SMPU instances for DMA_L2 and SPI flash address space.

SMPU Functional Description

The following sections provide details on the function of the SMPU module. If the region security settings allow transactions to go through, the ID in the ID-based region protection settings can still filter the transactions.

For the memory that an SMPU protects, programs can configure region-based settings with the `SMPU_RCTL[n]` registers. (There can be multiple SMPUs in a system.) The `SMPU_RCTL[n]` registers define the ID-based protection for memory regions.

If the target address does not reside in any configured memory region, the transaction permission resorts back to the global configuration setting.

Protection Units

Each SMPU provides two protection units, A and B for ID-based matching in the region-based memory protection. This feature provides a degree of flexibility for the user to match against multiple IDs.

Instruction Fetches

When the core executes instructions from memory, this operation is also considered a memory transaction. If the SMPU is configured to protect a memory region from read accesses that contain instructions, the core cannot fetch and execute these instructions.

Using Cache

When the processor uses both cache and the SMPU, there are a few issues to be aware of. If the SMPU is configured to protect a memory region from write accesses, instruction fetches from a core are still possible since instructions are not updated and replaced during run time.

NOTE: The debugger typically replaces an instruction with a breakpoint instruction for software breakpoints. If a memory region is protected against write accesses, software breakpoints are not possible unless the SMPU is configured with the appropriate system master ID of the debugger. The configuration allows it to perform a write-access.

In the case where memory is used for data, a read access or cache fill is not possible if the memory is blocked from read accesses. If read accesses are allowed but write accesses are disallowed, then there is an issue with coherency. The cache is filled but when the cache is updated and must be written back to the SMPU protected memory, the write-access is blocked.

In general, exercise caution when using both the SMPU and cache.

Speculative Reads

If speculative reads are enabled (`SMPU_CTL.RSDIS = 0`), the SMPU forwards the read transaction directly to the memory before checking the protection setting corresponding to the addressed memory region. This functionality saves one clock cycle in the clock domain of the SMPU. The SMPU checks the protection setting while the read

transaction occurs with the memory. If the protection setting dictates that the target memory address is blocked, the SMPU blocks the read to the master.

If speculative reads are disabled (`SMPU_CTL.RSDIS = 1`), the SMPU checks the protection settings first and forwards the transaction to memory only if it passes the configured protection settings. This functionality incurs a one-cycle latency per read.

NOTE: Reads affect certain memory operations such as automatic clearing of the memory (that is, FIFOs). When the SMPU protects this type of memory, disable read speculation since the blocking can occur without the read transaction reaching the target memory.

ADSP-2156x SMPU Register List

The System Memory Protection Unit (SMPU) provides selective protection of the processor's memory resources. The SMPU includes a set of processor events that can be monitored during program execution. A set of registers governs SMPU operations. For more information on SMPU functionality, see the SMPU register descriptions.

Table 11-2: ADSP-2156x SMPU Register List

Name	Description
<code>SMPU_BADDR</code>	Bus Error Address Register
<code>SMPU_BDTLS</code>	Bus Error Details Register
<code>SMPU_CTL</code>	SMPU Control Register
<code>SMPU_IADDR</code>	Interrupt Address Register
<code>SMPU_IDTLS</code>	Interrupt Details Register
<code>SMPU_RADDR[n]</code>	Region n Address Register
<code>SMPU_RCTL[n]</code>	Region n Control Register
<code>SMPU_REVID</code>	SMPU Revision ID Register
<code>SMPU_RIDA[n]</code>	Region n ID A Register
<code>SMPU_RIDB[n]</code>	Region n ID B Register
<code>SMPU_RIDMSKA[n]</code>	Region n ID Mask A Register
<code>SMPU_RIDMSKB[n]</code>	Region n ID Mask B Register
<code>SMPU_SECURECTL</code>	SMPU Control Secure Accesses Register
<code>SMPU_SECURERCTL[n]</code>	Region n Control Secure Accesses Register
<code>SMPU_STAT</code>	SMPU Status Register

SMPU Interrupts

The SMPU has one interrupt with the SEC ID = 187. See the [System Event Controller \(SEC\)](#) chapter for complete information on interrupt generation and use.

Memory Writes

A write transaction to address n is prevented when the following is true.

Address n is in memory region m and memory region m is write-protected ($\text{SMPU_RCTL}[n].\text{WPROTEN} = 1$) and ID is not a match. (See [ID Comparison](#)). The block occurs because the memory region is configured for write-protection and the ID comparison does not result in a match. If an ID comparison results in a match, the write transaction is allowed through.

Memory Reads

A read transaction from address n is prevented when the following is true:

- Address n is in memory region and memory region m is read-protected ($\text{SMPU_RCTL}[n].\text{RPROTEN} = 1$) and
- ID is not a match

The block occurs because the memory region is configured for read-protection and the ID comparison does not result in a match. If the ID comparison results in a match, the read transaction is permitted. (See [ID Comparison](#)).

ID Comparison

ID comparison automatically occurs during region-based memory protection. ID matches allow the transaction to bypass the configured memory protection for that region. The following sections describe the calculation of a write ID match and read ID match.

Write Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming write transaction:

- The $\text{SMPU_RCTL}[n].\text{WIDCINV}$ bit
- The $\text{SMPU_RIDA}[n].\text{ID}$ and $\text{SMPU_RIDB}[n].\text{ID}$ bit fields
- The $\text{SMPU_RIDMSKA}[n].\text{MSK}$ and $\text{SMPU_RIDMSKB}[n].\text{MSK}$ bit fields

Write IDA match = ((ID of incoming write transaction AND $\text{SMPU_RIDMSKA}[n].\text{MSK}$) == ($\text{SMPU_RIDA}[n].\text{ID}$ AND $\text{SMPU_RIDMSKA}[n].\text{MSK}$))

Write IDB match = ((ID of incoming write transaction AND $\text{SMPU_RIDMSKB}[n].\text{MSK}$) == ($\text{SMPU_RIDB}[n].\text{ID}$ AND $\text{SMPU_RIDMSKB}[n].\text{MSK}$))

Write ID match = (Write IDA match OR Write IDB match) XOR $\text{SMPU_RCTL}[n].\text{WIDCINV}$ bit

Read Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming read transaction:

- The `SMPU_RCTL[n].RIDCINV` bit
- The `SMPU_RIDA[n].ID` and `SMPU_RIDB[n].ID` bit fields
- The `SMPU_RIDMSKA[n].MSK` and `SMPU_RIDMSKB[n].MSK` bit fields

Read IDA match = ((ID of incoming read transaction AND `SMPU_RIDMSKA[n].MSK`) == (`SMPU_RIDA[n].ID` AND `SMPU_RIDMSKA[n].MSK`))

Read IDB match = ((ID of incoming read transaction AND `SMPU_RIDMSKB[n].MSK`) == (`SMPU_RIDB[n].ID` AND `SMPU_RIDMSKB[n].MSK`))

Read ID match = (Read IDA match OR Read IDB match) XOR `SMPU_RCTL[n].RIDCINV`

In the two cases described above, the incoming transaction (either write or read) ID is AND'ed with the configured mask value in protection unit A. It is then compared to the value of the configured ID value which is also AND'ed with the configured mask value in protection unit A. The mask provides a method to allow a group of IDs to match. This process is also performed for protection unit B. The two outcomes (from A and B) are then OR'ed together.

Depending on the setting of the `SMPU_RCTL[n].RIDCINV` or the `SMPU_RCTL[n].WIDCINV` bits, the ID match comparison is inverted or not. The final result after applying the inversion, `SMPU_RCTL[n].RIDCINV`, or `SMPU_RCTL[n].WIDCINV`, determines whether the transaction bypasses the protection.

Usage

The masks, `SMPU_RIDMSKA[n]` and `SMPU_RIDMSKB[n]`, are AND'ed with both the incoming transaction ID and the configured ID in `SMPU_RIDA[n].ID` and `SMPU_RIDB[n].ID`, respectively. By default the masks are zero. If ID-based region protection is enabled by setting the `SMPU_RCTL[n].WPROTEN` or `SMPU_RCTL[n].RPROTEN` bit fields and the masks are not set, the ID comparison essentially compares zeros. The comparison allows all transactions to bypass (if the region-based security setting is also configured in a way to allow transactions to go through for the region). To have the ID-based region protection to function, the mask registers and ID registers must also be set.

System IDs

The *System Master IDs* table provides the IDs for the system masters. An x means that the bit can be a 0 or a 1. There are multiple IDs associated with that particular system master.

Table 11-3: System Master IDs

ASIB Name	IID	SIID	ID
SP0A	0	0	11'b0000000x000
SP0B	0	1	11'b0000000x001
SP1A	0	2	11'b0000000x010
SP1B	0	3	11'b0000000x011

Table 11-3: System Master IDs (Continued)

ASIB Name	IID	SIID	ID
SP2A	0	4	11'b0000000x100
SP2B	0	5	11'b0000000x101
SP3A	0	6	11'b0000000x110
SP3B	0	7	11'b0000000x111
CRC0_CH0	1	6	11'b0010000x110
CRC0_CH1	1	7	11'b0010000x111
MLB	2	4	11'b00100000100
UART0_TX	3	0	11'b0011000x000
UART0_RX	3	4	11'b0011000x100
UART2_TX	3	3	11'b0011000x011
SPI0TX	4	0	11'b0100000x000
SPI0RX	4	1	11'b0100000x001
SPI1TX	4	2	11'b0100000x010
SPI1RX	4	3	11'b0100000x011
SPI2TX	4	5	11'b0100000x101
SPI2RX	4	4	11'b0100000x100
LP0	5	0	11'b0101000x000
UART1_TX	3	1	11'b0011000x001
UART1_RX	3	2	11'b0011000x010
LP1	5	1	11'b0101000x001
CRYPTO	6	2	11'b01100000010
ACC_CH0	6	0	11'b0110xxxx000
ACC_CH1	6	1	11'b0110xxxx001
EMDMA0_CH0	7	0	11'b01110000000
EMDMA0_CH1	7	1	11'b01110000001
EMDMA1_CH0	8	1	11'b10000000001
EMDMA1_CH1	8	0	11'b10000000000
MSMDMA_CH0	9	0	11'b1001000x000
MSMDMA_CH1	9	1	11'b1001000x001
DBG	9	2	11'b10010000010
ETR	9	3	11'b10010000011

Table 11-3: System Master IDs (Continued)

ASIB Name	IID	SIID	ID
CRC1_CH0	10	4	11'b1010000x100
CRC1_CH1	10	5	11'b1010000x101
UART2_RX	3	5	11'b0011000x101
SH0_DPORT	11	0	11'b1011xxxx000
SH0_IPORT	11	1	11'b10110000001
HSMDMA_CH0	12	0	11'b1100000x000
HSMDMA_CH1	12	1	11'b1100000x001
SP4A	13	0	11'b1101000x000
SP4B	13	1	11'b1101000x001
SP5A	13	2	11'b1101000x010
SP5B	13	3	11'b1101000x011
SP6A	13	4	11'b1101000x100
SP6B	13	5	11'b1101000x101
SP7A	13	6	11'b1101000x110
SP7B	13	7	11'b1101000x111
SH0_MMR	14	0	11'b1110xxxx000

Memory Region

Memory regions can start at address 0x00000000 or at any address that is a multiple of its size. The *Supported Memory Region Size and Alignment* table shows the memory region sizes that the processor supports and the alignment of the memory region. (X values are do-not-care).

SMPU supports a maximum of eight regions.

Table 11-4: Supported Memory Region Size and Alignment

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
4KB	0b00000	0xXXXXX000	-
8KB	0b00001	0xXXXXN000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
16KB	0b00010	0xXXXXN000	0x0, 0x4, 0x8, 0xC
32KB	0b00011	0xXXXXN000	0x0, 0x8
64KB	0b00100	0xXXXX0000	-
128KB	0b00101	0xXXXXN0000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE

Table 11-4: Supported Memory Region Size and Alignment (Continued)

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
256KB	0b00110	0xXXXN0000	0x0, 0x4, 0x8, 0xC
512KB	0b00111	0xXXXN0000	0x0, 0x8
1MB	0b01000	0xXXX00000	-
2MB	0b01001	0xXXN00000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
4MB	0b01010	0xXXN00000	0x0, 0x4, 0x8, 0xC
8MB	0b01011	0xXXN00000	0x0, 0x8
16MB	0b01100	0xXX000000	-
32MB	0b01101	0xXN000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
64MB	0b01110	0xXN000000	0x0, 0x4, 0x8, 0xC
128MB	0b01111	0xXN000000	0x0, 0x8
256MB	0b10000	0xX0000000	-
512MB	0b10001	0xN0000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
1GB	0b10010	0xN0000000	0x0, 0x4, 0x8, 0xC
2GB	0b10011	0xN0000000	0x0, 0x8
4GB	0b10100	0x00000000	-

For the case where the region size is selected as 4 GB, the region address must be at address 0x00000000.

NOTE: If a memory region address is not aligned to its size, the memory region start address protected by the SMPU is the configured address with the corresponding least significant bits masked. For example, if the size is configured for 16 KB (SMPU_RCTL[n].SIZE = 0b00010), and the base address is configured for SMPU_RADDR[n].BADDR = 0x00005018, the actual base address used by the SMPU is 0x00004000. When SMPU_RADDR[n].BADDR is read back, the program reads 0x00005000. This functionality is because only bits [11:0] are reserved as 0's. Programs must use care when setting the base address as it is not always the true base address.

SMPU Definitions

To make the best use of the SMPU, it is useful to understand the terms in this section.

Global Protection

Guarding of the entire memory space for the particular SMPU instantiation.

Region-Based Protection

Guarding individual segments of memory inside the memory space for the particular SMPU instantiation.

ID Match

A successful comparison of the ID associated with the incoming transaction and the ID and MASK configured in the SMPU.

SMPU Block Diagram

The *SMPU Top-Level Block Diagram* shows the SMPU block.

As seen in the diagram, the SMPU sits between the memory port (SCB master port) and the SCB fabric (SCB slave port). It acts as a gateway analyzing the transaction requests. It either rejects the transaction request or allows access based on the user-programmed configuration of the SMPU.

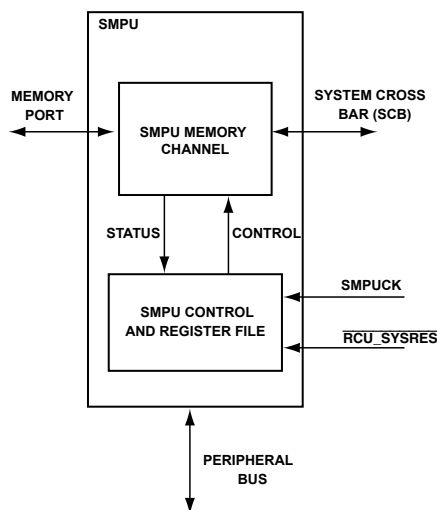


Figure 11-1: SMPU Top-Level Block Diagram

SMPU Architectural Concepts

The following sections provide brief descriptions of the architecture of the SMPU module.

Default Setting

At reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces.

Latency

The SMPU adds latency to all the transactions to the memory except reads when read speculation is enabled (`SMPU_CTL.RSDIS = 0`). In this case, read accesses are always forwarded to the memory and read responses are generated according to the SMPU settings. If read speculation is disabled (`SMPU_CTL.RSDIS = 1`), reads are blocked if they cause a security or protection violation. The SMPU generates the SCB read response that corresponds to a blocked transaction.

If read speculation is enabled, the SMPU adds 1 clock cycle latency to the read transaction. If read speculation is disabled, the SMPU adds 2 clock cycles latency to the read transaction.

SMPU Operating Modes

The SMPU does not have any strict modes of operation. However, it can be configured for region-based protection where a master with a particular ID can be blocked or allowed based on settings in the `SMPU_RCTL[n]` register.

Region-based protection is programmed with registers:

- `SMPU_RCTL[n]`
- `SMPU_RADDR[n]`
- `SMPU_RIDA[n]`
- `SMPU_RIDMSKA[n]`
- `SMPU_RIDB[n]`
- `SMPU_RIDMSKB[n]`

SMPU Interrupt Signals

There is one interrupt signal associated with the SMPU. If interrupts are enabled, the `SMPU_STAT . IRQ` bit is set. The `SMPU_IRQ` signal is asserted when the SMPU detects a memory access violation. The target address triggering the interrupt is found in the `SMPU_IADDR` register. The `SMPU_IDTLS` register provides further details about the cause of the interrupt.

Write errors are prioritized over read errors.

Protection violations (an ID-based violation) can trigger the SMPU interrupt, and can be enabled independently. The protection violation interrupt is enabled by setting the `SMPU_CTL . PINTEN` bit.

The SMPU interrupt is asserted for any of the following conditions:

If a second memory access violation occurs while the `SMPU_STAT . IRQ` bit is set, the `SMPU_STAT . IOVR` (interrupt overrun) bit is set. The `SMPU_IADDR` and the `SMPU_IDTLS` registers are not updated until the `SMPU_STAT . IRQ` bit is cleared. Any information on the subsequent interrupt is lost. Once the `SMPU_STAT . IRQ` bit and the `SMPU_STAT . IOVR` bit are cleared, any new memory access violations can trigger an interrupt and its details can be captured.

NOTE: When a blocked access occurs, the SMPU triggers an interrupt when interrupt generation is enabled. The SMPU can also be configured to generate a bus error that propagates back to the system master. The system master can also trigger an interrupt due to this bus error.

NOTE: On the processor, each SMPU instance has an interrupt. All of the SMPU interrupts are OR'ed and mapped to a single SMPU interrupt on the SEC. While servicing the SMPU interrupt, check all of the `SMPU_STAT` registers to determine which triggered the interrupt. The interrupt service routine clears the `SMPU_STAT . IRQ` bit of the all of the `SMPU_STAT` registers for which the interrupt is triggered.

SMPU Status and Error Signals

If bus errors are enabled (`SMPU_CTL.PBEDIS = 0`), the SMPU generates and returns a bus error to the master initiating the blocked access. This bit also sets the `SMPU_STAT.BERR` bit. The `SMPU_BADDR` and `SMPU_BDTLS` registers can be read to get the address and details of the transaction that caused the SMPU to generate the error.

Write errors are prioritized over read errors.

A bus error status is returned to the system master if:

- an ID-based violation happened and the `SMPU_CTL.PBEDIS` bit = 0

If a second memory access violation occurs while the `SMPU_STAT.BERR` bit is set, the `SMPU_STAT.BEOVR` bit (bus error overrun) is set. The `SMPU_BADDR` and the `SMPU_BDTLS` registers are not updated until the `SMPU_STAT.IRQ` bit is cleared. The information about the transaction that caused the `SMPU_STAT.BEOVR` bit to be set is lost.

NOTE: If both the protection violation interrupt is not enabled (`SMPU_CTL.PINTEN = 0`) and the protection bus error is disabled (`SMPU_CTL.PBEDIS = 1`), the SMPU blocks invalid transactions. However, it does not provide any status or interrupt information indicating that a transaction is blocked.

ADSP-2156x SMPU Register Descriptions

The System Memory Protection Unit (SMPU) contains the following registers.

Table 11-5: ADSP-2156x SMPU Register List

Name	Description
<code>SMPU_BADDR</code>	Bus Error Address Register
<code>SMPU_BDTLS</code>	Bus Error Details Register
<code>SMPU_CTL</code>	SMPU Control Register
<code>SMPU_IADDR</code>	Interrupt Address Register
<code>SMPU_IDTLS</code>	Interrupt Details Register
<code>SMPU_RADDR[n]</code>	Region n Address Register
<code>SMPU_RCTL[n]</code>	Region n Control Register
<code>SMPU_REVID</code>	SMPU Revision ID Register
<code>SMPU_RIDA[n]</code>	Region n ID A Register
<code>SMPU_RIDB[n]</code>	Region n ID B Register
<code>SMPU_RIDMSKA[n]</code>	Region n ID Mask A Register
<code>SMPU_RIDMSKB[n]</code>	Region n ID Mask B Register
<code>SMPU_SECURECTL</code>	SMPU Control Secure Accesses Register

Table 11-5: ADSP-2156x SMPU Register List (Continued)

Name	Description
SMPU_SECUREERCTL[n]	Region n Control Secure Accesses Register
SMPU_STAT	SMPU Status Register

Bus Error Address Register

Programs read the `SMPU_BADDR` and the `SMPU_BDTLS` registers to determine the cause of a bus error. Write errors are prioritized over read errors.

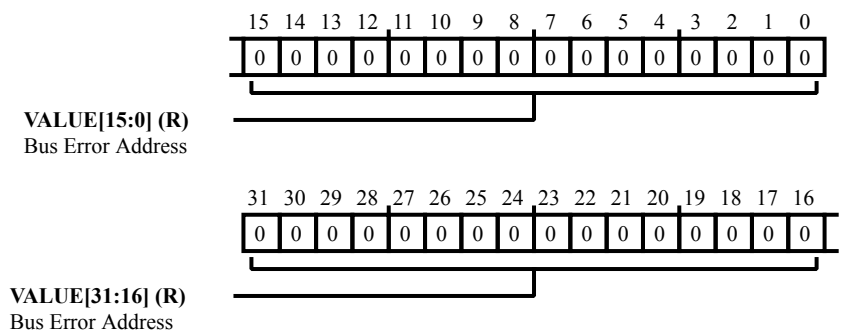


Figure 11-2: SMPU_BADDR Register Diagram

Table 11-6: SMPU_BADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bus Error Address. The <code>SMPU_BADDR.VALUE</code> bit field contains the address of the bus error.

Bus Error Details Register

The `SMPU_BDTLS` register indicates the ID of the bus error transaction, whether the transaction that caused the last bus error was a read, a write, secure or non-secure.

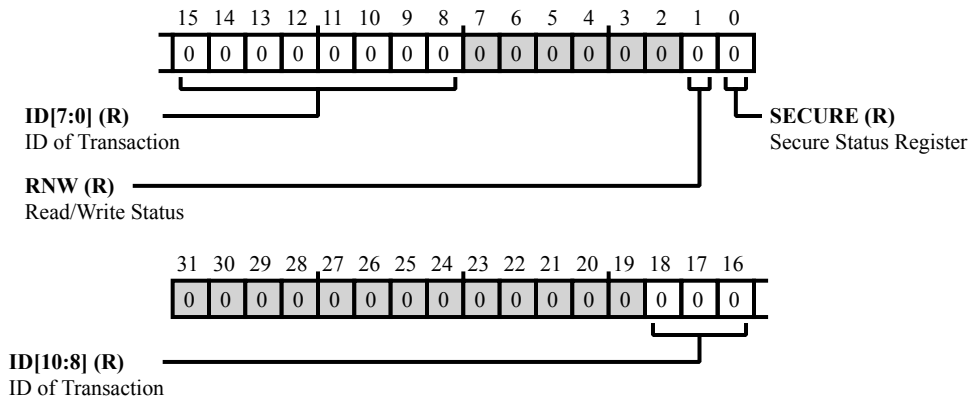


Figure 11-3: `SMPU_BDTLS` Register Diagram

Table 11-7: `SMPU_BDTLS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:8 (R/NW)	ID	ID of Transaction. The <code>SMPU_BDTLS . ID</code> bit field provides the ID of the transaction that caused the bad address error.
1 (R/NW)	RNW	Read/Write Status. The <code>SMPU_BDTLS . RNW</code> bit indicates whether the last transaction that caused the bad address error was a read or write.
	0	Transaction that caused last bus error was a write
	1	Transaction that caused last bus error was a read
0 (R/NW)	SECURE	Secure Status Register. The <code>SMPU_BDTLS . SECURE</code> bit indicates whether the last transaction that caused the bad address error was secure or non-secure.
	0	Transaction that caused last bus error was non-secure
	1	Transaction that caused last bus error was secure

SMPU Control Register

The `SMPU_CTL` register provides access to the locking control, error interrupts and SMPU violations.

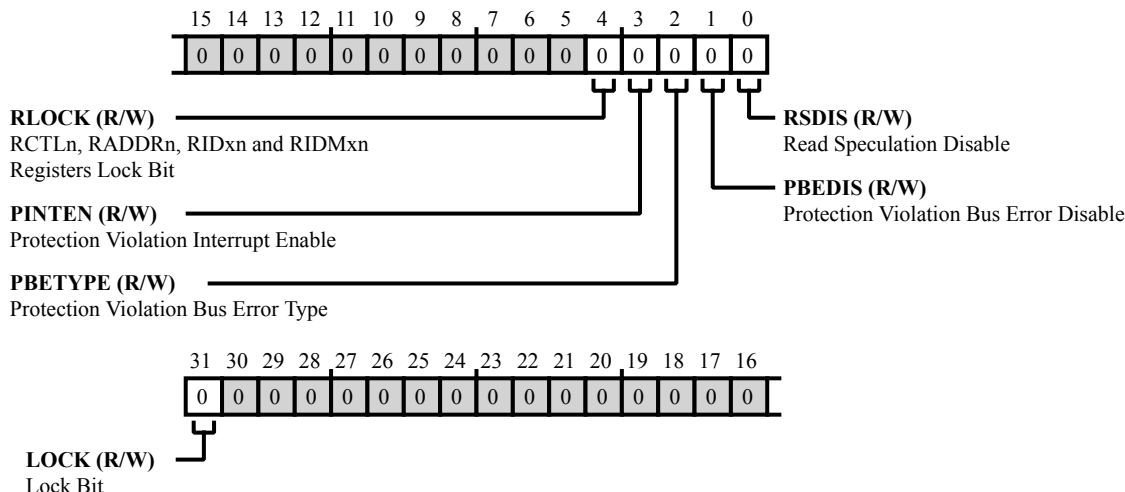


Figure 11-4: SMPU_CTL Register Diagram

Table 11-8: SMPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit. When the <code>SMPU_CTL</code> .LOCK bit is set and the global lock signal is asserted from the SPU, the <code>SMPU_CTL</code> register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.
		0 CTL Global Lock Disable. The <code>SMPU_CTL</code> register is not write-protected.
		1 CTL Global Lock Enable. The <code>SMPU_CTL</code> register is write-protected.
4 (R/W)	RLOCK	RCTLn, RADDRn, RIDxn and RIDMxn Registers Lock Bit. When the <code>SMPU_CTL</code> .RLOCK bit is set, all the registers associated with region-based control (<code>SMPU_RCTL[n]</code> , <code>SMPU_RADDR[n]</code> , <code>SMPU_RIDA[n]</code> , <code>SMPU_RIDB[n]</code> , <code>SMPU_RIDMSKA[n]</code> and <code>SMPU_RIDMSKB[n]</code>) are write-protected when the global lock signal is active from the SPU. Write access is allowed again when the global lock signal is deasserted.
		0 Region Registers Write-Protect Enable. All region registers are not write-protected.
		1 Region Registers Write-Protect Disable. All region registers are write-protected.

Table 11-8: SMPU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	PINTEN	Protection Violation Interrupt Enable. The SMPU_CTL.PINTEN bit controls whether or not an interrupt is generated when a protection violation occurs.
		0 Protection Violation IRQ Disable. The protection violation interrupt is disabled.
		1 Protection Violation IRQ Enable. The protection violation interrupt is enabled.
2 (R/W)	PBETYPE	Protection Violation Bus Error Type. The SMPU_CTL.PBETYPE bit controls whether a protection violation produces a decode error or a slave error.
		0 Decode Error Type. Decode error for transactions that violate the configured protection.
		1 Slave Error Type. Slave Error for transactions which violate the configured protection
1 (R/W)	PBEDIS	Protection Violation Bus Error Disable. If set, the SMPU_CTL.PBEDIS bit blocks protection violations, but does not cause a bus error.
		0 Bus Error Generation Enable. Transactions which violate the configured protection are blocked and cause a bus error.
		1 Bus Error Generation Disable. Transactions which violate the configured protection are blocked but do not cause a bus error.
0 (R/W)	RSDIS	Read Speculation Disable. The SMPU_CTL.RSDIS bit controls whether or not the read addresses are checked before being sent to the slave.
		0 Read Speculation Enable. Read addresses are sent to the slave without checking.
		1 Read Speculation Disable. Read addresses are checked before being sent to the slave.

Interrupt Address Register

The `SMPU_IADDR` register indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The SMPU issues a bus error for this condition.

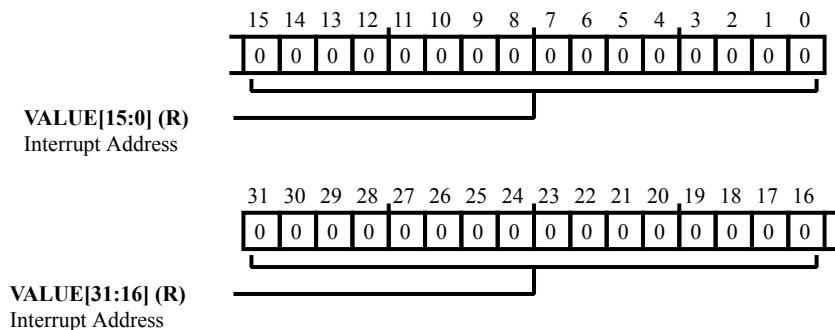


Figure 11-5: `SMPU_IADDR` Register Diagram

Table 11-9: `SMPU_IADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Interrupt Address. The <code>SMPU_IADDR.VALUE</code> bit field is the address where an attempt to access an unimplemented address or a non-aligned access has occurred.

Interrupt Details Register

The `SMPU_IDTLS` register provides the ID of the last signaled interrupt, whether the interrupt was caused by a read or write, and whether the transaction that caused the last signaled interrupt was secure.

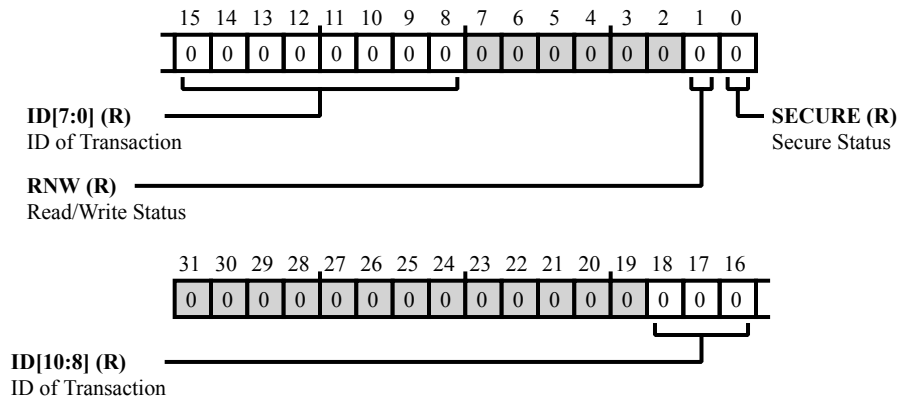


Figure 11-6: `SMPU_IDTLS` Register Diagram

Table 11-10: `SMPU_IDTLS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:8 (R/NW)	ID	ID of Transaction. The <code>SMPU_IDTLS.ID</code> bit field provides the ID of the transaction that caused the interrupt.
1 (R/NW)	RNW	Read/Write Status. The <code>SMPU_IDTLS.RNW</code> bit indicates whether the last transaction that caused the interrupt was a read or write.
		0 Transaction that caused last signaled interrupt was a write
		1 Transaction that caused last signaled interrupt was a read
0 (R/NW)	SECURE	Secure Status. The <code>SMPU_IDTLS.SECURE</code> bit indicates whether the last transaction that caused the interrupt was secure or non-secure.
		0 Transaction that caused last signaled interrupt was non-secure
		1 Transaction that caused last signaled interrupt was secure

Region n Address Register

The `SMPU_RADDR[n]` register is used to define the base address for a memory region to be protected.

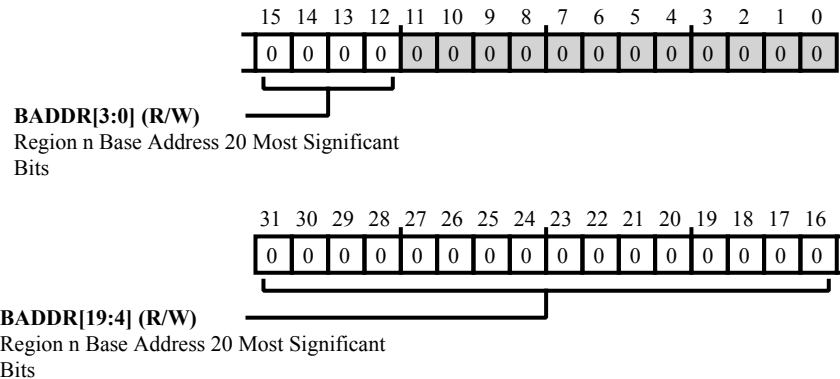


Figure 11-7: `SMPU_RADDR[n]` Register Diagram

Table 11-11: `SMPU_RADDR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:12 (R/W)	BADDR	Region n Base Address 20 Most Significant Bits. The <code>SMPU_RADDR[n].BADDR</code> bit field defines the base address for a memory region to be protected.

Region n Control Register

The `SMPU_RCTL[n]` register is used to define the level of protection for a region of memory. The protection of a region is controlled and defined by this register and the `SMPU_RADDR[n]`, `SMPU_RIDA[n]`, `SMPU_RIDB[n]`, `SMPU_RIDMSKA[n]`, and `SMPU_RIDMSKB[n]` registers.

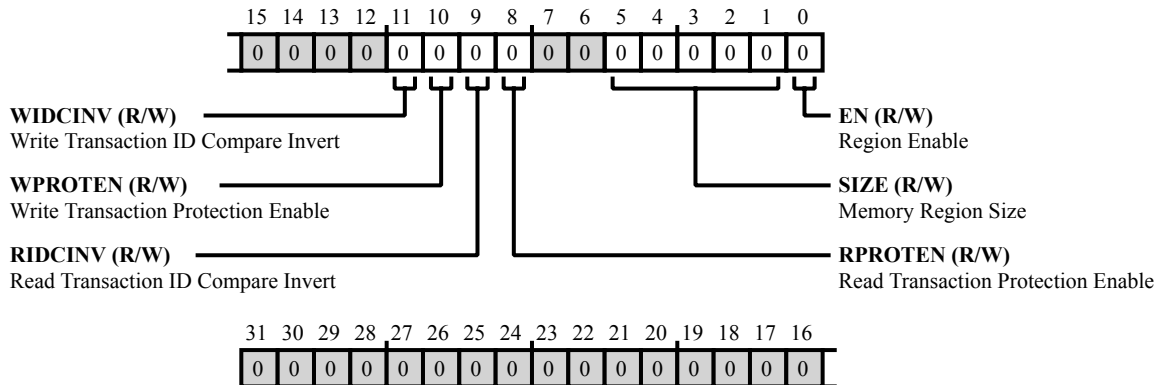


Figure 11-8: `SMPU_RCTL[n]` Register Diagram

Table 11-12: `SMPU_RCTL[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	WIDCINV	Write Transaction ID Compare Invert. The <code>SMPU_RCTL[n].WIDCINV</code> bit inverts the write ID match result.
		0 Write transaction ID comparison result not inverted
		1 Write transaction ID comparison result inverted
10 (R/W)	WPROTEN	Write Transaction Protection Enable. The <code>SMPU_RCTL[n].WPROTEN</code> bit enables protection against ID-based write transactions for the memory region.
		0 Write transaction ID-based protection disabled
		1 Write transaction ID-based protection enabled
9 (R/W)	RIDCINV	Read Transaction ID Compare Invert. When the <code>SMPU_RCTL[n].RIDCINV</code> bit is set, the read ID match result is inverted.
		0 Read transaction ID comparison result not inverted
		1 Read transaction ID comparison result inverted

Table 11-12: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	RPROTEN	Read Transaction Protection Enable. The SMPU_RCTL[n].RPROTEN bit enable bit to turn on protection against ID-based read transactions for the memory region.
		0 Read transaction ID-based protection disabled
		1 Read transaction ID-based protection enabled
5:1 (R/W)	SIZE	Memory Region Size. The SMPU_RCTL[n].SIZE bit defines the size of the memory region to be protected.
		0 4 KB
		1 8 KB
		2 16 KB
		3 32 KB
		4 64 KB
		5 128 KB
		6 256 KB
		7 512 KB
		8 1 MB
		9 2 MB
		10 4 MB
		11 8 MB
		12 16 MB
		13 32 MB
		14 64 MB
		15 128 MB
		16 256 MB
		17 512 MB
		18 1 GB
		19 2 GB
		20 4 GB
21-31	Reserved	

Table 11-12: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Region Enable. The SMPU_RCTL[n].EN bit enables the protection of a region.	
		0	Disabled
		1	Enabled

SMPU Revision ID Register

The `SMPU_REVID` register provides the major and minor revision numbers of this module.

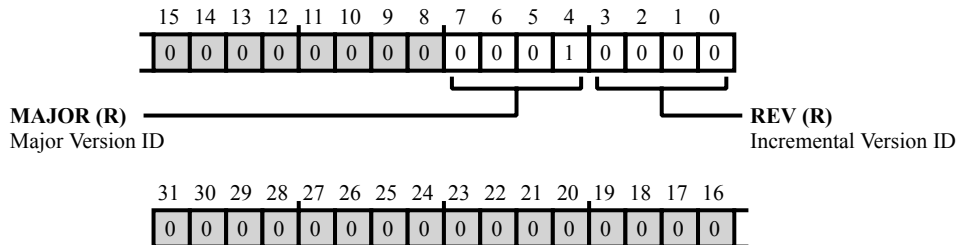


Figure 11-9: SMPU_REVID Register Diagram

Table 11-13: SMPU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Region n ID A Register

The `SMPU_RIDA[n]` register is used for ID comparison 'A'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more detail, refer to the ID Comparison section.

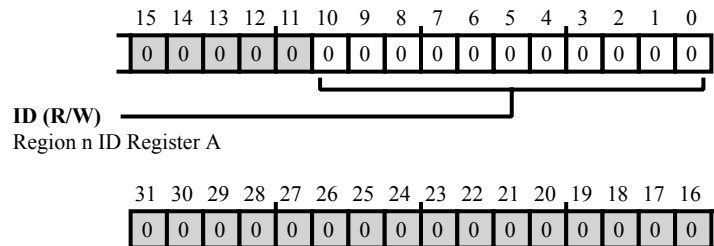


Figure 11-10: `SMPU_RIDA[n]` Register Diagram

Table 11-14: `SMPU_RIDA[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	ID	Region n ID Register A. The <code>SMPU_RIDA[n].ID</code> bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID B Register

The `SMPU_RIDB[n]` register is used for ID comparison 'B'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more details, refer to the ID Comparison section.

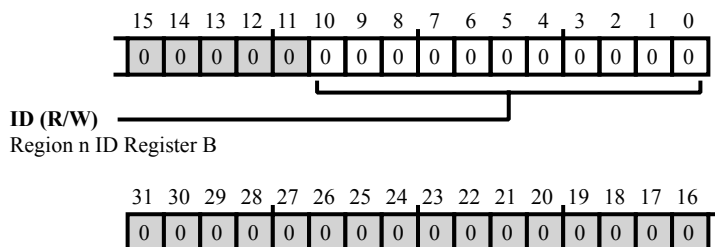


Figure 11-11: SMPU_RIDB[n] Register Diagram

Table 11-15: SMPU_RIDB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	ID	Region n ID Register B. The <code>SMPU_RIDB[n].ID</code> bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID Mask A Register

The `SMPU_RIDMSKA[n]` register is used for ID comparison 'A'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

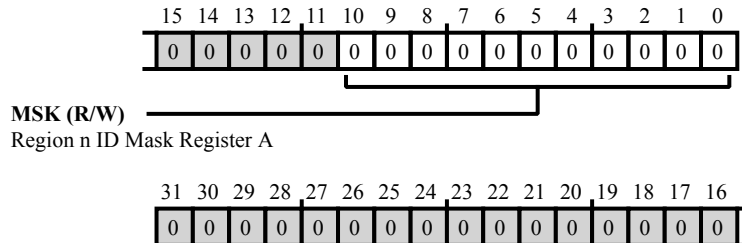


Figure 11-12: `SMPU_RIDMSKA[n]` Register Diagram

Table 11-16: `SMPU_RIDMSKA[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MSK	Region n ID Mask Register A. The <code>SMPU_RIDMSKA[n].MSK</code> bit field, combined with the incoming transaction, provides the means to bypass the configured memory protection for a region.

Region n ID Mask B Register

The `SMPU_RIDMSKB[n]` register is used for ID comparison 'B'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

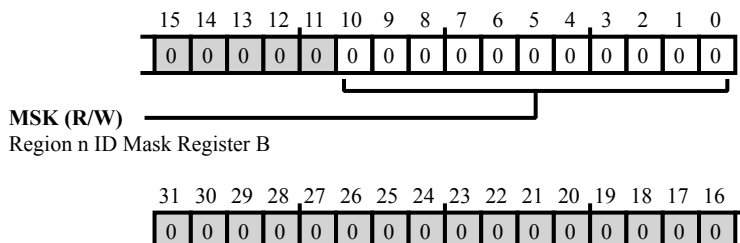


Figure 11-13: SMPU_RIDMSKB[n] Register Diagram

Table 11-17: SMPU_RIDMSKB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	MSK	Region n ID Mask Register B. The <code>SMPU_RIDMSKB[n].MSK</code> bit field, combined with the incoming transaction provides the means to bypass the configured memory protection for a region.

SMPU Control Secure Accesses Register

The `SMPU_SECURECTL` register provides the bits required to set up the security settings for the processor. These settings includes error generation and read/write security.

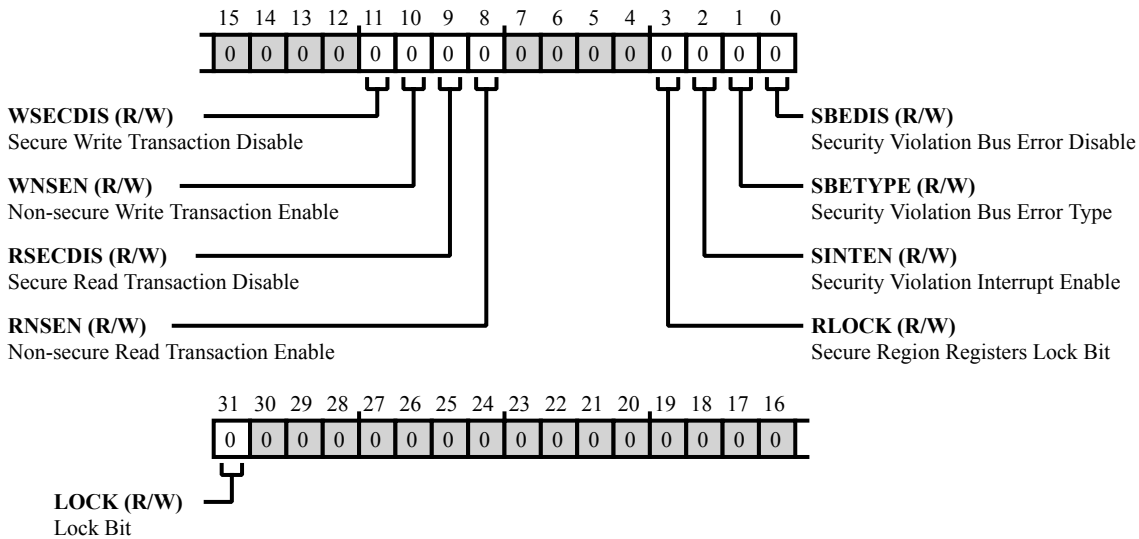


Figure 11-14: `SMPU_SECURECTL` Register Diagram

Table 11-18: `SMPU_SECURECTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit. When the <code>SMPU_SECURECTL</code> .LOCK bit is set and the global lock signal is asserted from the SPU, the <code>SMPU_SECURECTL</code> register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.
		0 <code>SMPU_SECURECTL</code> is not write-protected
		1 <code>SMPU_SECURECTL</code> is write-protected
11 (R/W)	WSECDIS	Secure Write Transaction Disable. The <code>SMPU_SECURECTL</code> .WSECDIS bit disables secure write transactions.
		0 Enable secure write transactions
		1 Disable secure write transactions
10 (R/W)	WNSEN	Non-secure Write Transaction Enable. The <code>SMPU_SECURECTL</code> .WNSEN bit enables non-secure write transactions.
		0 Disable non-secure writes
		1 Enable non-secure writes

Table 11-18: SMPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	RSECDIS	Secure Read Transaction Disable. The SMPU_SECURECTL.RSECDIS bit disables secure read transactions.
		0 Enable secure read transactions
		1 Disable secure read transactions
8 (R/W)	RNSEN	Non-secure Read Transaction Enable. The SMPU_SECURECTL.RNSEN bit enables non-secure read transactions.
		0 Disable non-secure read transactions
		1 Enable non-secure read transactions
3 (R/W)	RLOCK	Secure Region Registers Lock Bit. When the SMPU_SECURECTL.RLOCK bit is set, the secure region control registers, SMPU_SECURECTL[n], are write-protected when the global lock signal is active from the SPU. When the global lock signal is deasserted, write access is allowed again.
		0 Disable write-protection on secure region registers
		1 Enable write-protection on secure region registers
2 (R/W)	SINTEN	Security Violation Interrupt Enable. The SMPU_SECURECTL.SINTEN bit enables interrupt generation when a security violation occurs.
		0 Disable security settings violation interrupt
		1 Enable security settings violation interrupt
1 (R/W)	SBETYPE	Security Violation Bus Error Type. The SMPU_SECURECTL.SBETYPE bit controls whether a decode error or a slave error is returned when a security violation occurs.
		0 Return a decode error error which violates the security settings
		1 Return a slave error which violates the security settings
0 (R/W)	SBEDIS	Security Violation Bus Error Disable. The SMPU_SECURECTL.SBEDIS bit controls whether or not a bus error is caused when a security violation occurs.
		0 Enable bus error
		1 Disable bus error

Region n Control Secure Accesses Register

The `SMPU_SECURERCTL[n]` register contains bits that configure read/write security for a specific region.

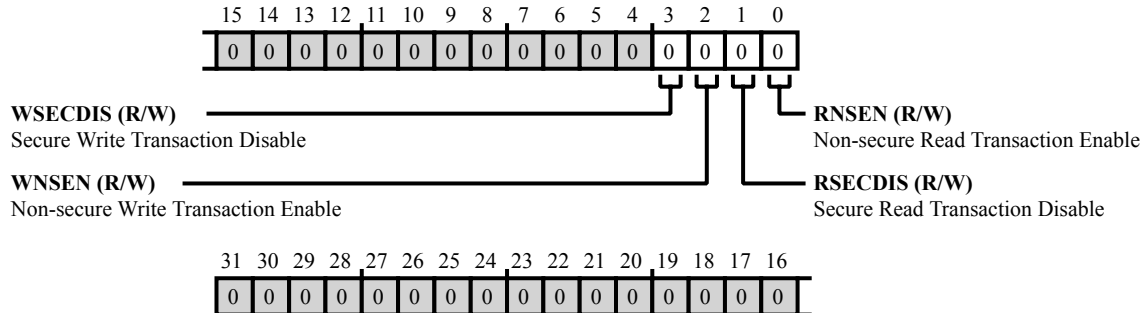


Figure 11-15: SMPU_SECURERCTL[n] Register Diagram

Table 11-19: SMPU_SECURERCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	WSECDIS	Secure Write Transaction Disable. The <code>SMPU_SECURERCTL[n].WSECDIS</code> bit disables secure write transactions for the memory region.
		0 Enable secure write transactions to this region
		1 Disable secure write transactions to this region
2 (R/W)	WNSEN	Non-secure Write Transaction Enable. This <code>SMPU_SECURERCTL[n].WNSEN</code> bit enables non-secure write transactions for the memory region.
		0 Disable non-secure write transactions to this region
		1 Enable non-secure write transactions to this region
1 (R/W)	RSECDIS	Secure Read Transaction Disable. The <code>SMPU_SECURERCTL[n].RSECDIS</code> bit disables secure read transactions for the memory region.
		0 Enable secure read transactions to this region
		1 Disable secure read transactions to this region
0 (R/W)	RNSEN	Non-secure Read Transaction Enable. The <code>SMPU_SECURERCTL[n].RNSEN</code> bit enables non-secure read transactions for the memory region.
		0 Disable non-secure read transactions to this region
		1 Enable non-secure read transactions to this region

SMPU Status Register

The `SMPU_STAT` register provides the state of the SMPU and indicates various errors. All bits in this register are write 1 to clear.

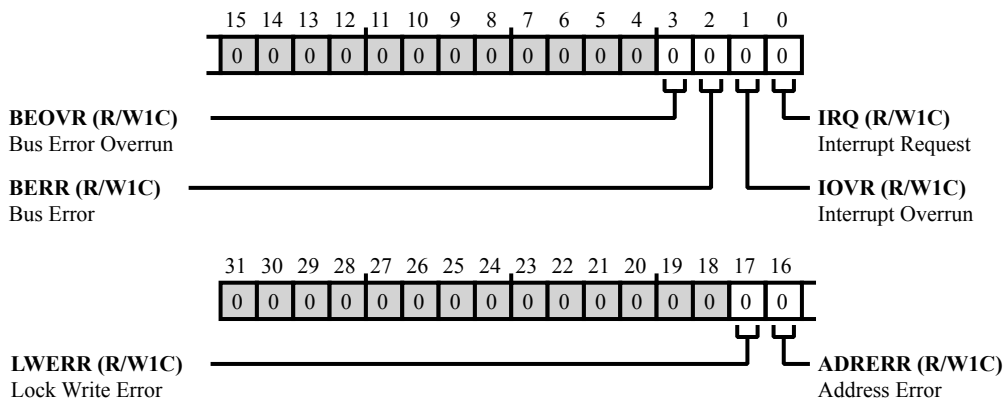


Figure 11-16: SMPU_STAT Register Diagram

Table 11-20: SMPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W1C)	LWERR	Lock Write Error. The <code>SMPU_STAT.LWERR</code> bit is set when <code>SMPU_CTL.LOCK</code> bit =1, the global lock signal is asserted from the SPU and a read or write attempt was made to the <code>SMPU_CTL</code> MMR.
		0 No Lock Write Error
		1 Lock Write Error
16 (R/W1C)	ADRERR	Address Error. The <code>SMPU_STAT.ADRERR</code> bit is set when the SMPU MMR is accessed as an un-aligned address, or when a read-only MMR is written to.
		0 No Address Error
		1 Address Error
3 (R/W1C)	BEOVR	Bus Error Overrun. The <code>SMPU_STAT.BEOVR</code> bit indicates that another bus error had occurred. Any new information about the most recent violation which caused the bus error is not captured.
		0 No Bus Error overrun
		1 Bus Error overrun has occurred

Table 11-20: SMPU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	BERR	Bus Error. This SMPU_STAT . BERR bit indicates if a bus error was generated.
		0 No Bus Error since this bit has been cleared
		1 Bus Error has been generated
1 (R/W1C)	IOVR	Interrupt Overrun. The SMPU_STAT . IOVR bit indicates if another violation occurred while the previous violation interrupt was not finished being serviced. Information about the most recent violation is then not captured.
		0 No Interrupt overrun
		1 Interrupt overrun has occurred
0 (R/W1C)	IRQ	Interrupt Request. The SMPU_STAT . IRQ bit provides an indication that an interrupt has been generated.
		0 No Interrupt since this bit has been cleared
		1 Interrupt has been generated

12 General-Purpose Ports (PORT)

This section describes general-purpose ports, pin multiplexing, general-purpose input/output (GPIO) functionality, and pin interrupts. The general-purpose ports provide the following three functions:

- Pin multiplexing scheme
- GPIO functionality
- Pin interrupt requests

NOTE: In this chapter, the naming convention for registers and bits omits the alphabetic group enumeration to refer to any and all of the ports. For example, `PORT_FER` represents registers `PORTA_FER`, `PORTB_FER`, and so on. Likewise `PORT_FER.PX1` represents bits `PA1`, `PB1`, and so on.

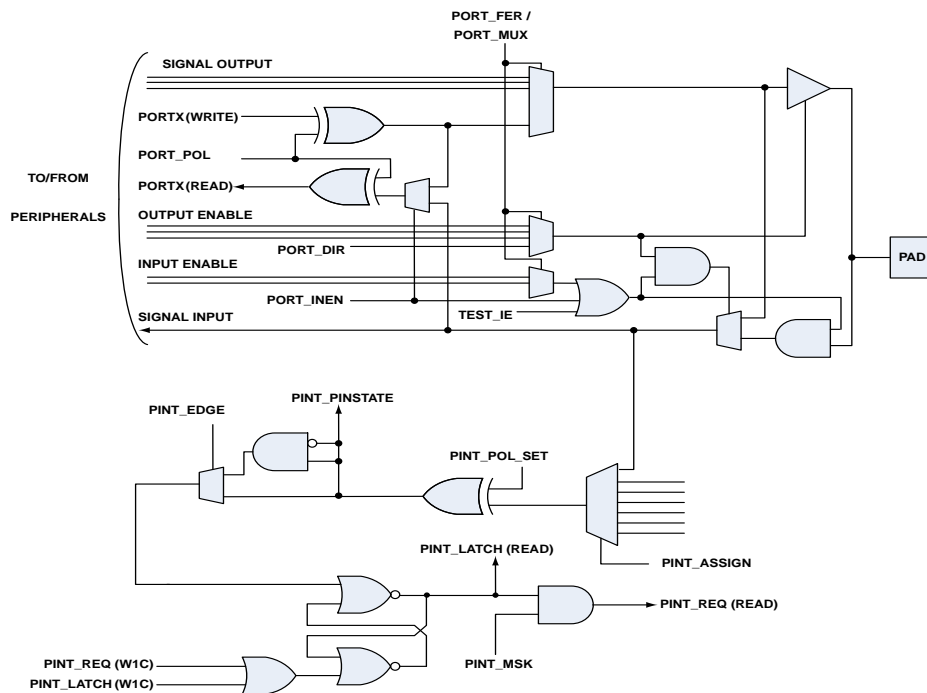


Figure 12-1: Simplified GPIO and Pin Interrupt Signal Flow

PORT Features

The PORTs include the following features:

- Input mode, output mode, and open-drain mode of GPIO operation
- Port multiplexing controlled on a pin-by-pin basis
- No external glue hardware required for unused pins
- All port pins provide interrupt request functionality
- Byte-wide pin-to-interrupt request assignment

PORT Functional Description

The number of ports and each's composition are defined in the processor datasheet. Each port has a dedicated set of MMR registers that control pin functions and operates in general-purpose I/O (GPIO) mode by default, as controlled by the port-specific `PORT_FER` register. Each bit in this register, as well as the other PORT MMRs, represents a specific GPIO pin on the specified port.

Input Mode, Output Mode, and Open-Drain Mode of GPIO Operation

At reset, every GPIO pin defaults to input mode with the input drivers disabled. To enable any GPIO input driver, set the bits corresponding to the individual pins in the appropriate input enable register (`PORT_INEN`).

The GPIO output drivers are enabled by setting the corresponding bits in the direction registers (`PORT_DIR`).

The PORT can use every GPIO in open-drain mode by clearing the respective bit in the `PORT_DATA` register or setting the respective bit in the `PORT_DATA_CLR` register. Then, set the corresponding bit in the `PORT_INEN` register. Read from the `PORT_DATA` register to obtain the status from the pin.

Port Multiplexing Controlled on Pin-by-Pin Basis

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (`PORT_FER`) registers and the 32-bit port multiplexing (`PORT_MUX`) registers.

All Port Pins Provide Interrupt Functionality

Pin interrupts are completely decoupled from GPIO functionality. Pins are connected to the system event controller (SEC) via the PINTx modules, each of which is configurable in terms of which port pins are sensed for interrupt generation.

ADSP-2156x PORT Register List

The PORT module (PORT) regulates the use of the multiplexable processor pins. Every port pin can operate in general-purpose I/O (GPIO) mode or as an alternate function. This GPIO operation is the default after processor reset and is controlled by a set of registers that control GPIO functionality. Every bit in these registers represents a certain GPIO pin of a specific port. For more information on PORT functionality, see the PORT register descriptions.

Table 12-1: ADSP-2156x PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_TRIG_TGL	Port x GPIO Trigger Toggle Register

ADSP-2156x PORT Trigger List

Table 12-2: ADSP-2156x PORT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
		None	

Table 12-3: ADSP-2156x PORT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
111	PORTA_TOGGLE	PORTA Port Toggle Trigger	Pulse
112	PORTB_TOGGLE	PORTB Port Toggle Trigger	Pulse
113	PORTC_TOGGLE	PORTC Port Toggle Trigger	Pulse

ADSP-2156x PINT Register List

The Pin Interrupt module (PINT) controls the pin-to-interrupt assignment in a byte-wide manner. The pin-interrupt assignment registers do not consist of 32 individual bits. They consist of four control bytes, each functioning as a multiplexer control. For more information, see the PINT register descriptions.

All PINT registers are 32 bits wide and can be accessed by 32-bit load/store instructions. They also support 16-bit operation where the upper 16 bits are ignored and the application uses the lower 16 bits only. Consequently, all PINT registers support 32-bit accesses as well as 16-bit accesses for the lower half words. Applications may use faster 16-bit accesses as long as they do not require functionality of upper register halves.

Table 12-4: ADSP-2156x PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

ADSP-2156x PINT Interrupt List

Table 12-5: ADSP-2156x PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
26	PINT0_BLOCK	PINT0 Pin Interrupt Block 0	Level	
27	PINT1_BLOCK	PINT1 Pin Interrupt Block 1	Level	
28	PINT2_BLOCK	PINT2 Pin Interrupt Block 2	Level	

ADSP-2156x PINT Trigger List

Table 12-6: ADSP-2156x PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
13	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
14	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level
15	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level

Table 12-7: ADSP-2156x PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-2156x PADS Register List

The PADS controls signal hysteresis and other system interface signal features for a number of module interfaces.

Table 12-8: ADSP-2156x PADS Register List

Name	Description
PADS_DAI0_0_DS	DAI0 1 to 10 pins DS control
PADS_DAI0_1_DS	DAI0 11 to 20 pins DS control
PADS_DAI0_IE	DAI0 Port Input Enable Control Register
PADS_DAI1_0_DS	DAI1 1 to 10 pins DS control
PADS_DAI1_1_DS	DAI1 11 to 20 pins DS control
PADS_DAI1_IE	DAI1 Port Input Enable Control Register
PADS_DAI[n]_PDE	DAIx Pull-Down Enable
PADS_DAI[n]_PUE	DAIx Pull-Up Enable
PADS_NONPORTS_DS	Non-GPIO Drive Strength Register
PADS_PCFG0	Peripheral PAD Configuration0 Register
PADS_PORTA0_DS	PORTA 0 to 7 pins DS control
PADS_PORTA1_DS	PORTA 8 - 15 pins DS control
PADS_PORTB0_DS	PORTB 0 to 7 pins DS control
PADS_PORTB1_DS	PORTB 8 - 15 pins DS control
PADS_PORTC0_DS	PORTC 0 to 7 pins DS control
PADS_PORT[n]_PDE	PORTx Pull-Down Enable
PADS_PORT[n]_PUE	PORTx Pull-Up Enable

PORT Architectural Concepts

These sections describe in more detail how the PORT module connects externally to pins and internally to the MMR bus. Ports are named alphabetically beginning with A.

- [Internal Interfaces](#)
- [External Interfaces](#)
- [GPIO Pin Function](#)
- [Port Multiplexing Control](#)

Internal Interfaces

All of the pin multiplexing, GPIO, and pin interrupt control block MMRs can be accessed through the MMR bus. There is no DMA support. Each of the pin interrupt (PINTx) modules has its own dedicated interrupt request output signal that connects directly to the system event controller (SEC).

External Interfaces

The pin multiplexing hardware can be seen as a layer between the on-chip peripherals and the silicon pads connecting to the physical pins/balls or the package, as controlled by the PORT unit.

GPIO Pin Function

By default, the PORT sets every GPIO pin to input mode. The input drivers are not enabled, which avoids the need for unnecessary current sinks and external termination resistors on unused pins.

Input Mode

The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the bits corresponding to the PORT pins in the appropriate input enable register ([PORT_INEN](#)). When enabled, a read from the [PORT_DATA](#) register returns the logical state of the input pins. However, the input signal does not overwrite the state of the internal flip-flop used for providing output to the same pin. Only software can alter the state. If the input driver is enabled, a write to the [PORT_DATA](#) register can alter the state of the flip-flop, but the change cannot be read back.

Output Mode

Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the bits corresponding to the PORT pins in the appropriate direction register. The PORT implements direction registers as a pair of write-1-to-set (W1S) and write-1-to-clear (W1C) MMRs called [PORT_DIR_SET](#) and [PORT_DIR_CLR](#), respectively. As such, software can alter the direction of the signal flow on individual GPIO pins without impacting other GPIOs on the same port.

Both the [PORT_DIR_SET](#) and [PORT_DIR_CLR](#) registers return the same value when read, and a logical 1 indicates an enabled output. The [PORT_DATA](#) registers control the state of output pins. A logical 0 drives the output low while a logical 1 drives the output high.

While writes to the [PORT_DATA](#) register can alter all of the GPIOs on a specific port at once, there are also a pair of W1S and W1C MMRs called [PORT_DATA_SET](#) and [PORT_DATA_CLR](#), respectively. These registers enable

the manipulation of individual GPIO outputs. The state of the outputs can be obtained by reading the `PORT_DATA` registers. Because the state of the GPIO output can be controlled before the output driver is enabled, set or clear the internal flip-flop first by programming this register to avoid volatile levels on the output pin.

Trigger Toggle Mode

Any GPIO pin that has been configured for output mode can be toggled using a trigger input from the Trigger Routing Unit (TRU). To enable this functionality for a particular GPIO, set the appropriate bit in the `PORT_TRIG_TGL` register. Any subsequent trigger for the designated port causes all GPIO outputs set in the `PORT_TRIG_TGL` register to toggle.

To avoid unpredictable behavior, do not set, clear, or toggle the `PORT_DATA`, `PORT_DATA_SET`, `PORT_DATA_CLR`, or `PORT_DATA_TGL` registers when the GPIO output has the corresponding `PORT_TRIG_TGL` bit set. To change the state of the GPIO output using one of these registers, first clear the corresponding bit in the `PORT_TRIG_TGL` register.

Open-Drain Mode

Every GPIO can also be used in open-drain mode. First, either clear the respective bit in the `PORT_DATA` register or set the respective bit in the `PORT_DATA_CLR` register. Then, set the appropriate bit in the `PORT_INEN` register. Read from the `PORT_DATA` register to return the status from the pin rather than the state of the internal flip-flop.

By toggling the output driver through the `PORT_DIR_SET` and `PORT_DIR_CLR` register pair, the output signal can be pulled low or three-stated, as required. The polarity of the driven signal can be inverted when the internal flip-flop is set. When using a GPIO port in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pins.

Port Multiplexing Control

To configure pins properly, consult the processor datasheet to determine which bits in the `PORT_FER` and `PORT_MUX` register map to the pin of interest, and then set these registers appropriately for the desired function.

After reset, all port pins default to GPIO input mode with their output and input drivers disabled. As a result, all unused port pins can be left unconnected. Disabled pins appear as high-impedance to external circuits.

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (`PORT_FER`) registers and the 32-bit port multiplexing (`PORT_MUX`) registers.

The function enable register specifies whether the pin is used as a GPIO pin or allocated for use by a specific peripheral, but it does not specify what the peripheral function is. Each bit in the 16-bit `PORT_FER` register corresponds to an individual port pin. For example, if bit 1 (PA1) of the `PORTA_FER` register is cleared, the PA_01 pin is configured as a GPIO. When set, one of the available peripheral functions becomes active on the PA_01 pin instead.

Pairs of bits in the `PORT_MUX` register control the multiplexing between the peripheral functions available to an individual pin, as some PORT pins provide up to four possible peripheral functions.

Refer to the Signal Muxing table in the datasheet for the specific `PORT_MUX` settings.

PORT Event Control

The following sections describe event generation in the PORT module.

PORT Interrupt Signals

The pin interrupts are decoupled from GPIO functionality, providing the following advantages.

- Flexible mapping scheme enables pins from up to four different ports to be grouped into one common interrupt scheme.
- Interrupt requests work on input and output pins regardless of whether the pin is functioning as a GPIO or a peripheral.

The processor has a number of interrupt channels dedicated to pin interrupts, managed by a set of pin interrupt (PINT_x) blocks. Each PINT_x block can sense up to 32 GPIO pins, as described in the following list and figure.

- PINT0 can sense pin activity on PORTA and PORTB
- PINT1 can sense pin activity on PORTB and PORTC
- PINT2 can sense pin activity on PORTC

The processor supports both 32-bit and 16-bit peripheral bus accesses to PINT_x registers.

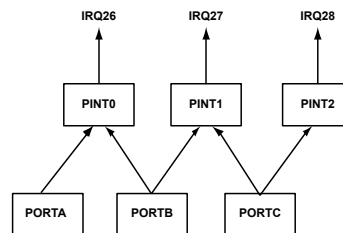


Figure 12-2: GPIO to PINT_x Assignment

Pins connect to the PINT_x module and then to the system event controller (SEC), as shown in the *PINT_x Block Diagram*.

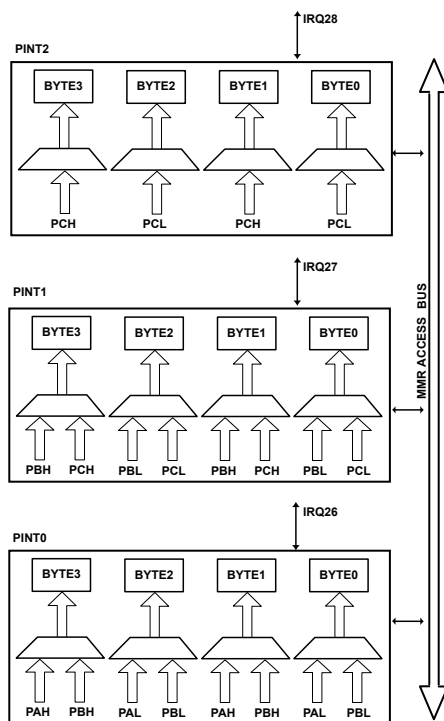


Figure 12-3: PINTx Block Diagram

As shown in the *PINTx Block Diagram*, each port is subdivided into two 8-pin half ports, upper (P×H) and lower (P×L). The `PINT_ASSIGN` registers control the 8-bit multiplexers associated with these half ports, where the lower half units (eight pins) can be forwarded to either byte 0 or byte 2 of the PINTx blocks, and the upper half units (eight pins) can be forwarded to either byte 1 or byte 3 of the PINTx blocks.

When a half port is assigned to a byte in any PINTx block, the state of the eight pins appears in the `PINT_PINSTATE` register, regardless of whether the pin is enabled for GPIO or peripheral functions (input or output). When neither the input nor output drivers of the pin are enabled, the pin state is read as zero. The `PINT_PINSTATE` register reports the inverted state of the pin when the `PINT_INV_SET` register activates the signal inverter. The inverter can be enabled on an individual bit-by-bit basis. Each bit in the `PINT_INV_SET/PINT_INV_CLR` register pair represents a pin signal.

By default, PORT interrupt request generation is level-sensitive, and an interrupt request occurs when the enabled pin is sensed as active high. Use the `PINT_EDGE_SET` register to change the interrupt request generation scheme to instead be edge-sensitive (rising edge generates the interrupt request). Use the `PINT_INV_SET` register to invert the polarity such that the PINTx block generates the interrupt request on active-low signals or falling edges.

The PINTx modules also assist when both signal edges must generate unique interrupt requests. If two different interrupt requests are required, the `PINT_ASSIGN` registers can route a single input signal to two different PINTx blocks, where one block inverts the signal in the `PINT_INV_SET` register and the other one does not. In this fashion, a unique software routine is associated with the hardware PINTx block that is generating the unique interrupt request for each signal edge. When both signal edges can be serviced by the same interrupt request, each half port can be routed to two separate bytes within a single PINTx block using the `PINT_ASSIGN` register, and then one of

the half ports needs to have the inversion enabled in the `PINT_INV_SET` register. The servicing software routine can then detect from the `PINT_LATCH` register whether a falling, rising, or both edges have occurred.

Regardless of whether level-sensitive or edge-sensitive mode is used, the hardware always latches an interrupt request. Latched signals can be read from the `PINT_LATCH` registers. Only a software or hardware reset clears the latches. To clear the latch, a W1C operation must be performed to the `PINT_REQ` or `PINT_LATCH` register. When in level-sensitive mode, the interrupt request remains asserted if the pin state does not change by the time the interrupt service routine exits.

Because every PINTx block groups up to 32 pin signals, the `PINT_MSK_SET/ PINT_MSK_CLR` register pair can control which of the signals can request an interrupt at the system level. Software can interrogate the `PINT_REQ` register for signaling pins. The `PINT_REQ` bits represent a logical AND between the mask and the latch. When any of these bits is set, the block output interrupt request is asserted.

PORT Programming Model

The *GPIO Programming Model Flow (Part 1)*, *GPIO Programming Model Flow (Part 2)*, and *GPIO Programming Model Flow (Part 3)* figures show the programming model for the general-purpose ports. This programming includes the GPIO input and output operation, open-drain mode, and the pin interrupt PINTx modules.

NOTE: These process flow diagrams connect where call-out letters appear. For example, call-out A in the *GPIO Programming Model Flow (Part 1)* diagram connects to call-out A in the *GPIO Programming Model Flow (Part 2)* diagram.

The following flowcharts describe the processes for setting up pins for various functions. Begin the process from the start label in the *GPIO Programming Model Flow (Part 1)* figure. The first decision (GPIO or peripheral) determines how to program the `PORT_FER` register. Set the bit(s) corresponding to the pin(s) to 1 to enable peripheral functionality or to 0 to enable GPIO mode. For more information on setting up for peripheral functions, refer to [Port Multiplexing Control](#).

If the pin is to be a GPIO pin, a subsequent series of decisions must be made that will impact how the `PORT_DATA`, `PORT_POL`, `PORT_DIR`, and `PORT_INEN` configuration registers must be programmed. Depending on the type of GPIO pin desired, some configurations do not apply and can have different meanings. The following paragraphs briefly describe the function of the different settings for each of the pin functions in the input, output, and open-drain GPIO modes. It is a best practice to use the SET or CLR versions of the PORT registers, where applicable, to effect changes on a pin-by-pin basis rather than on the full port.

For output mode, first clear the `PORT_DATA` register to set all the pins low. Then write the `PORT_DIR` register to define the direction of each pin (set the bits associated with the desired output pins to 1). In output mode, the other registers are not significant. The *GPIO Programming Model Flow (Part 1)* chart shows this flow starting at label 2.

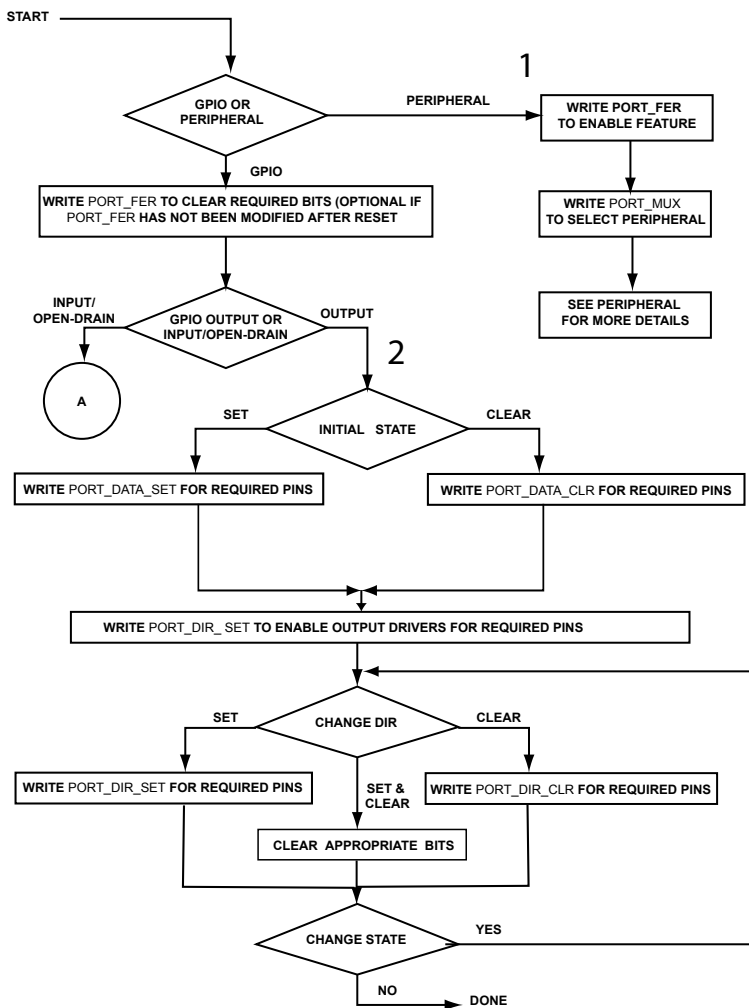


Figure 12-4: GPIO Programming Model Flow (Part 1)

For input mode, first decide the polarity for each pin using the `PORT_POL` register. Program the `PORT_DIR` register to define the appropriate pins as inputs (write a 0 to the bit location associated with the pin). If interrupt requests are desired, configure the PINT module as shown in the *GPIO Programming Model Flow (Part 3)* figure starting at label B. Finally, write the `PORT_INEN` register to enable the associated input drivers. The *GPIO Programming Model Flow (Part 2)* chart shows this entire flow starting at label 3.

For open-drain mode, set all pins low by clearing the `PORT_DATA` register. Then, use the `PORT_INEN` register to enable the appropriate input drivers. Set the `PORT_DIR` register in this mode to indicate whether the pin is in an active state or not (active being 0). The *GPIO Programming Model Flow (Part 2)* chart shows this flow starting at label 4.

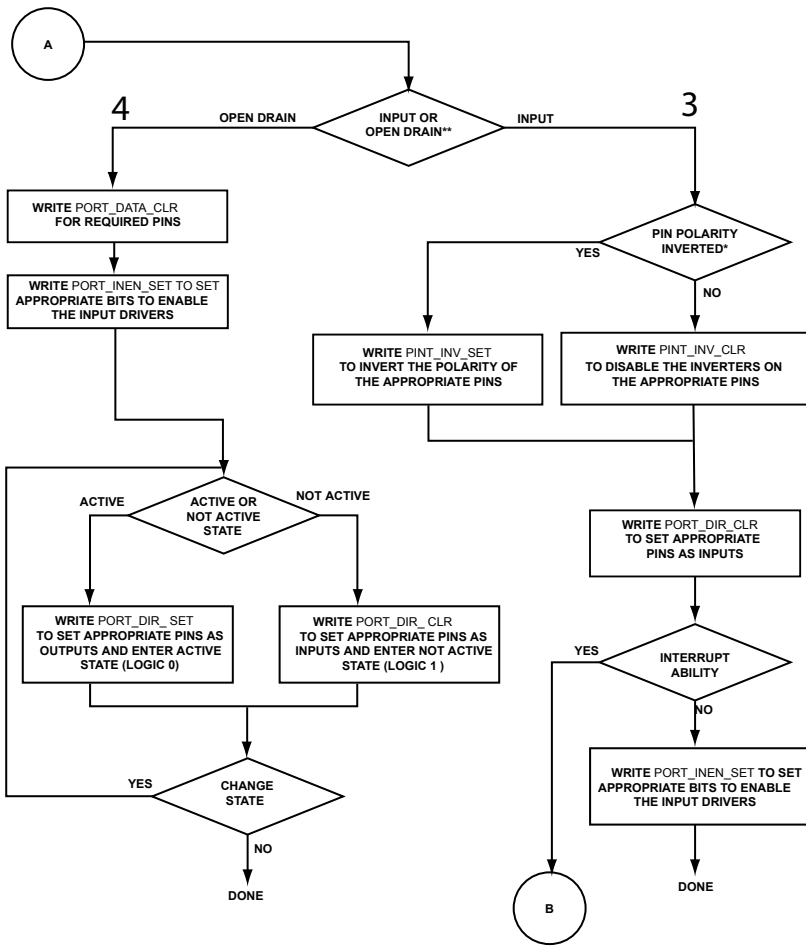


Figure 12-5: GPIO Programming Model Flow (Part 2)

Table 12-9: ADSP-2156x PORT Register List (Continued)

Name	Description
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_TRIG_TGL	Port x GPIO Trigger Toggle Register

Port x GPIO Data Register

The operation of the `PORT_DATA` register depends on whether the bit/pin is in output mode or input mode. In both modes, a set bit in the `PORT_DATA` register corresponds to a signal high on a GPIO pin. A cleared bit in the `PORT_DATA` register corresponds to a signal low on a GPIO pin.

The `PORT_DATA`, `PORT_DATA_SET`, and `PORT_DATA_CLR` registers control the state of GPIO pins in output mode. To enable output mode (and output drivers), use the `PORT_DIR_SET` and `PORT_DIR_CLR` registers.

Writes to the `PORT_DATA` register affect the state of all pins of the port that are in output mode. To set or clear specific pins without impacting other pins of the port, use the `PORT_DATA_SET` and `PORT_DATA_CLR` registers.

When the GPIO pins are in input mode (input driver is enabled with the `PORT_INEN` register), reads from the `PORT_DATA`, `PORT_DATA_SET`, and `PORT_DATA_CLR` registers return the state of the respective GPIO pins.

Note that when the input driver is not enabled, reads from the `PORT_DATA`, `PORT_DATA_SET`, and `PORT_DATA_CLR` registers return the value previously written to the registers.

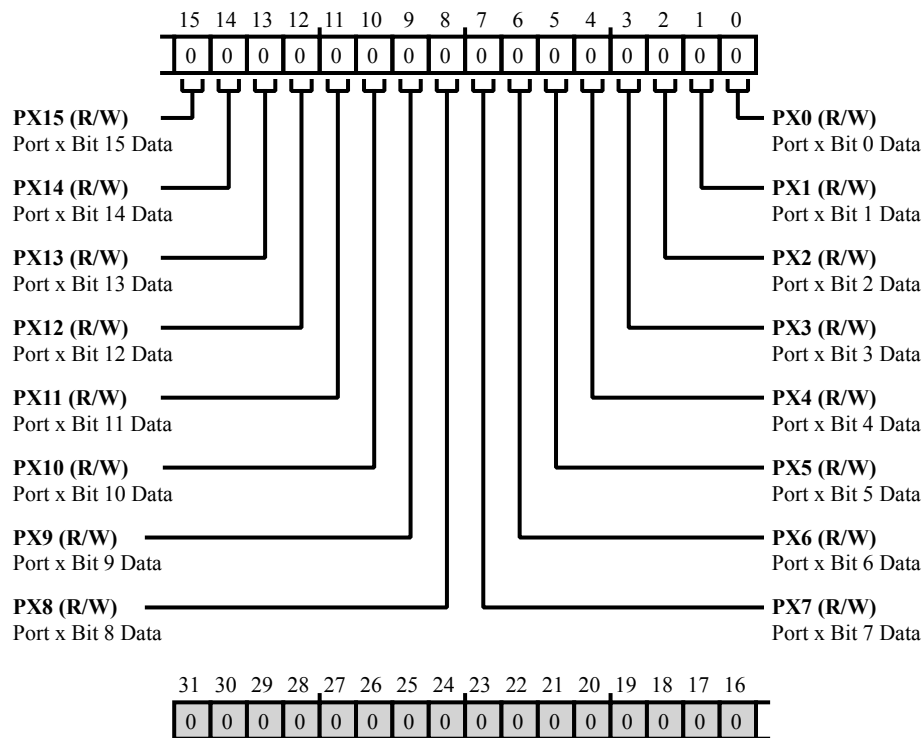


Figure 12-7: `PORT_DATA` Register Diagram

Table 12-10: PORT_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	Port x Bit 15 Data. The PORT_DATA.PX15 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
14 (R/W)	PX14	Port x Bit 14 Data. The PORT_DATA.PX14 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
13 (R/W)	PX13	Port x Bit 13 Data. The PORT_DATA.PX13 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
12 (R/W)	PX12	Port x Bit 12 Data. The PORT_DATA.PX12 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
11 (R/W)	PX11	Port x Bit 11 Data. The PORT_DATA.PX11 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
10 (R/W)	PX10	Port x Bit 10 Data. The PORT_DATA.PX10 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
9 (R/W)	PX9	Port x Bit 9 Data. The PORT_DATA.PX9 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High

Table 12-10: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	PX8	Port x Bit 8 Data. The PORT_DATA.PX8 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
7 (R/W)	PX7	Port x Bit 7 Data. The PORT_DATA.PX7 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
6 (R/W)	PX6	Port x Bit 6 Data. The PORT_DATA.PX6 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
5 (R/W)	PX5	Port x Bit 5 Data. The PORT_DATA.PX5 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
4 (R/W)	PX4	Port x Bit 4 Data. The PORT_DATA.PX4 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
3 (R/W)	PX3	Port x Bit 3 Data. The PORT_DATA.PX3 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
2 (R/W)	PX2	Port x Bit 2 Data. The PORT_DATA.PX2 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High

Table 12-10: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	PX1	Port x Bit 1 Data. The PORT_DATA.PX1 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High
0 (R/W)	PX0	Port x Bit 0 Data. The PORT_DATA.PX0 bit indicates a signal on a GPIO pin.
		0 Signal Low
		1 Signal High

Port x GPIO Data Clear Register

The `PORT_DATA_CLR` register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the `PORT_DATA` register description.

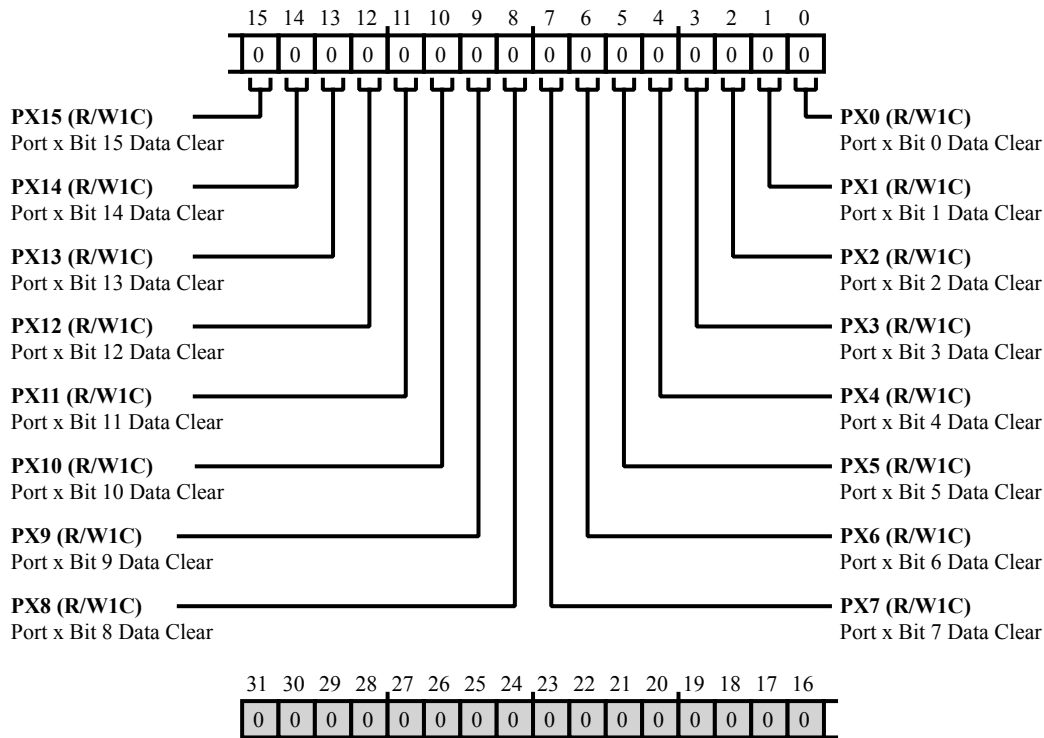


Figure 12-8: `PORT_DATA_CLR` Register Diagram

Table 12-11: `PORT_DATA_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PX15	Port x Bit 15 Data Clear. The <code>PORT_DATA_CLR</code> . PX15 bit clears the pin without impacting other pins of the port.
		0 No Effect
		1 Clear Bit. Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear. The <code>PORT_DATA_CLR</code> . PX14 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.

Table 12-11: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	PX13	Port x Bit 13 Data Clear. The PORT_DATA_CLR.PX13 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
12 (R/W1C)	PX12	Port x Bit 12 Data Clear. The PORT_DATA_CLR.PX12 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
11 (R/W1C)	PX11	Port x Bit 11 Data Clear. The PORT_DATA_CLR.PX11 bit clears the pin without impacting other pins of the port.
		0 No Effect
		1 Clear Bit. Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear. The PORT_DATA_CLR.PX10 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
9 (R/W1C)	PX9	Port x Bit 9 Data Clear. The PORT_DATA_CLR.PX9 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
8 (R/W1C)	PX8	Port x Bit 8 Data Clear. The PORT_DATA_CLR.PX8 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.

Table 12-11: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	PX7	Port x Bit 7 Data Clear. The PORT_DATA_CLR.PX7 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
6 (R/W1C)	PX6	Port x Bit 6 Data Clear. The PORT_DATA_CLR.PX6 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
5 (R/W1C)	PX5	Port x Bit 5 Data Clear. The PORT_DATA_CLR.PX5 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
4 (R/W1C)	PX4	Port x Bit 4 Data Clear. The PORT_DATA_CLR.PX4 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
3 (R/W1C)	PX3	Port x Bit 3 Data Clear. The PORT_DATA_CLR.PX3 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
2 (R/W1C)	PX2	Port x Bit 2 Data Clear. The PORT_DATA_CLR.PX2 bit clears the pin without impacting other pins of the port.
		0 No Effect Write 0 has no effect in output mode.
		1 Clear Bit Write 1 for signal low in output mode.

Table 12-11: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	PX1	Port x Bit 1 Data Clear. The PORT_DATA_CLR.PX1 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.
0 (R/W1C)	PX0	Port x Bit 0 Data Clear. The PORT_DATA_CLR.PX0 bit clears the pin without impacting other pins of the port.
		0 No Effect. Write 0 has no effect in output mode.
		1 Clear Bit. Write 1 for signal low in output mode.

Port x GPIO Data Set Register

The `PORT_DATA_SET` register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the `PORT_DATA` register description.

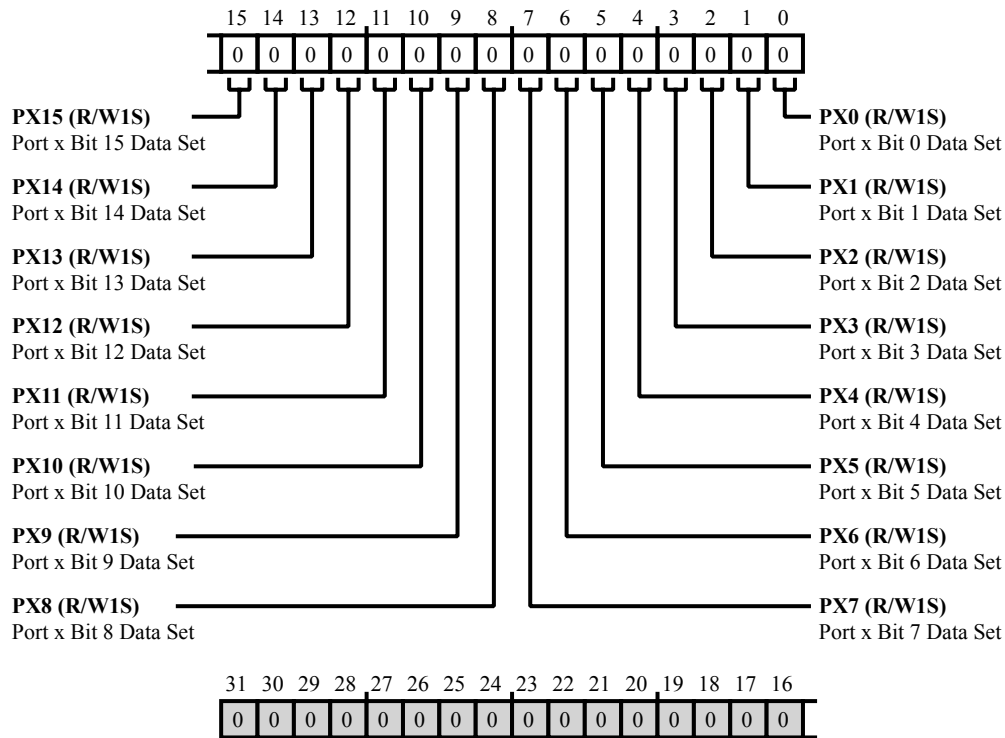


Figure 12-9: `PORT_DATA_SET` Register Diagram

Table 12-12: `PORT_DATA_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PX15	Port x Bit 15 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
14 (R/W1S)	PX14	Port x Bit 14 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
13 (R/W1S)	PX13	Port x Bit 13 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.

Table 12-12: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W1S)	PX12	Port x Bit 12 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
11 (R/W1S)	PX11	Port x Bit 11 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit.
10 (R/W1S)	PX10	Port x Bit 10 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
9 (R/W1S)	PX9	Port x Bit 9 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
8 (R/W1S)	PX8	Port x Bit 8 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
7 (R/W1S)	PX7	Port x Bit 7 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
6 (R/W1S)	PX6	Port x Bit 6 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
5 (R/W1S)	PX5	Port x Bit 5 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
4 (R/W1S)	PX4	Port x Bit 4 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
3 (R/W1S)	PX3	Port x Bit 3 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.

Table 12-12: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1S)	PX2	Port x Bit 2 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
1 (R/W1S)	PX1	Port x Bit 1 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.
0 (R/W1S)	PX0	Port x Bit 0 Data Set.
		0 No Effect. Write 0 has no effect in output mode.
		1 Set Bit. Write 1 for signal high in output mode.

Port x GPIO Output Toggle Register

The `PORT_DATA_TGL` register permits toggling the state of output GPIO pins. Setting bits in the `PORT_DATA_TGL` register affects the state of specific pins without impacting other pins of the port.

Reading the `PORT_DATA_TGL` returns the state of the `PORT_DATA` register output pin state, but does not return the input pin/signal state.

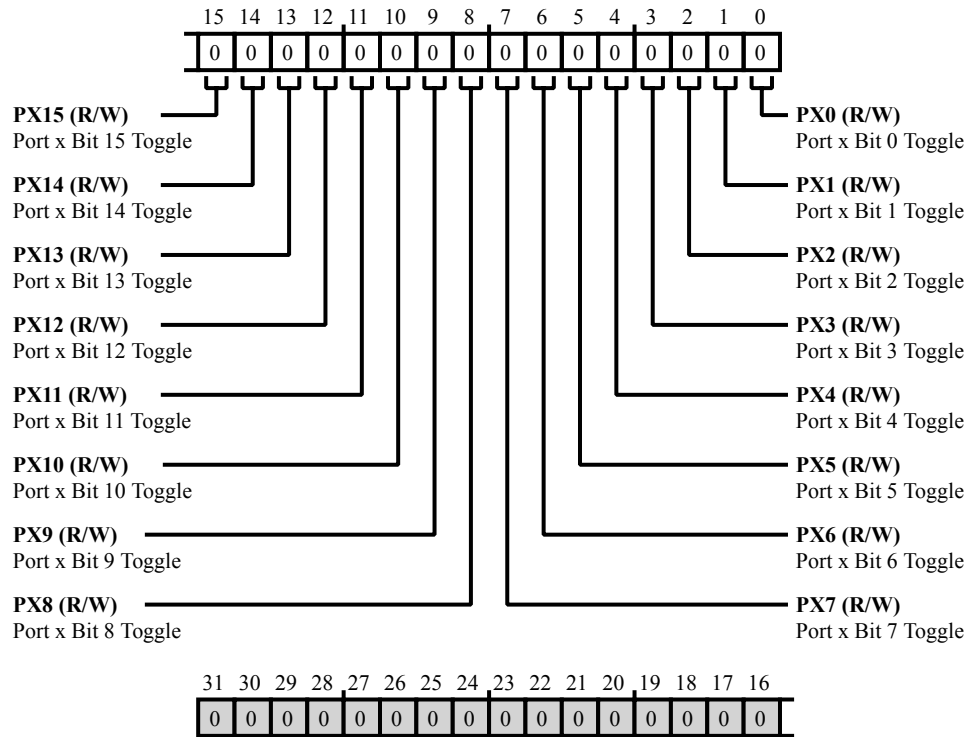


Figure 12-10: PORT_DATA_TGL Register Diagram

Table 12-13: PORT_DATA_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	Port x Bit 15 Toggle. The <code>PORT_DATA_TGL</code> . PX15 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit.
14 (R/W)	PX14	Port x Bit 14 Toggle. The <code>PORT_DATA_TGL</code> . PX14 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit

Table 12-13: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	PX13	Port x Bit 13 Toggle. The PORT_DATA_TGL.PX13 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
12 (R/W)	PX12	Port x Bit 12 Toggle. The PORT_DATA_TGL.PX12 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
11 (R/W)	PX11	Port x Bit 11 Toggle. The PORT_DATA_TGL.PX11 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
10 (R/W)	PX10	Port x Bit 10 Toggle. The PORT_DATA_TGL.PX10 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
9 (R/W)	PX9	Port x Bit 9 Toggle. The PORT_DATA_TGL.PX9 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
8 (R/W)	PX8	Port x Bit 8 Toggle. The PORT_DATA_TGL.PX8 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
7 (R/W)	PX7	Port x Bit 7 Toggle. The PORT_DATA_TGL.PX7 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit

Table 12-13: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	PX6	Port x Bit 6 Toggle. The PORT_DATA_TGL.PX6 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
5 (R/W)	PX5	Port x Bit 5 Toggle. The PORT_DATA_TGL.PX5 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
4 (R/W)	PX4	Port x Bit 4 Toggle. The PORT_DATA_TGL.PX4 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
3 (R/W)	PX3	Port x Bit 3 Toggle. The PORT_DATA_TGL.PX3 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
2 (R/W)	PX2	Port x Bit 2 Toggle. The PORT_DATA_TGL.PX2 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
1 (R/W)	PX1	Port x Bit 1 Toggle. The PORT_DATA_TGL.PX1 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit
0 (R/W)	PX0	Port x Bit 0 Toggle. The PORT_DATA_TGL.PX0 bit toggles the output GPIO bit/pin state.
		0 No Effect
		1 Toggle Bit

Port x GPIO Direction Register

The `PORT_DIR`, `PORT_DIR_SET`, and `PORT_DIR_CLR` registers select output or input mode for GPIO pins and enable output drivers. Use the `PORT_INEN`, `PORT_INEN_SET`, and `PORT_INEN_CLR` registers to enable or disable input drivers.

Writes to the `PORT_DIR` register affect the state of all pins of the port. To select a direction for specific pins without impacting other pins of the port, use the `PORT_DIR_SET` and `PORT_DIR_CLR` registers.

Setting a bit in the `PORT_DIR` register enables output mode on the corresponding a GPIO pin. Clearing a bit in the `PORT_DIR` register disables output mode on the corresponding GPIO pin.

Input Mode - The default mode of every GPIO pin after reset is the input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the corresponding bits in the `PORT_INEN` register. When enabled, a read from the `PORT_DATA` register returns the logical state of the input pin. The input signal does not overwrite the state of the bit used for the output case. That state can only be altered by software. If the input driver is enabled, a write to the `PORT_DATA` register can alter the state of the bit, but the change cannot be read back.

Output Mode - Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the corresponding bits in the `PORT_DIR`, `PORT_DIR_SET`, or `PORT_DIR_CLR` registers. By using the `PORT_DIR_SET` and `PORT_DIR_CLR` registers, the direction of the signal flow of individual GPIO pins can be altered by separate software threads without mutually impacting other GPIOs on the same port. Both registers return the same value when read. Because the state of the GPIO output can already be controlled before the output driver is enabled, it is recommended to first set or clear the bit (using the `PORT_DATA`, `PORT_DATA_SET`, or `PORT_DATA_CLR` registers) to avoid any volatile levels on the output.

Open-Drain Mode - Every GPIO can also be used in open-drain mode. To accomplish this, first, clear the respective bit in the `PORT_DATA` or `PORT_DATA_CLR` register. Then, set the one bit in the `PORT_INEN` register. Reads from the `PORT_DATA` register then return the status from the pin and do not return the state of the internal flip-flop. By toggling the output driver through the `PORT_DIR_SET` and `PORT_DIR_CLR` register pair, the output signal can be pulled low or three-stated as required. Note that the polarity of the driven signal can be inverted when the internal flip-flop is set instead. When a GPIO port is used in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pin.

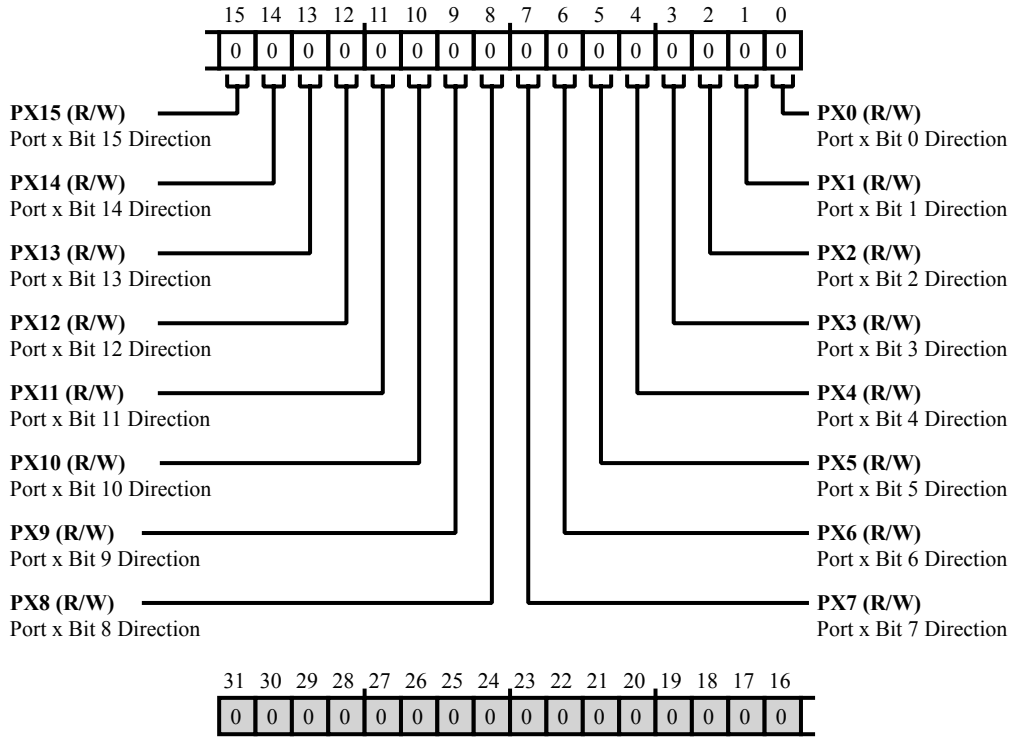


Figure 12-11: PORT_DIR Register Diagram

Table 12-14: PORT_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	Port x Bit 15 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.
14 (R/W)	PX14	Port x Bit 14 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.
13 (R/W)	PX13	Port x Bit 13 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.
12 (R/W)	PX12	Port x Bit 12 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.

Table 12-14: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	PX11	Port x Bit 11 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
10 (R/W)	PX10	Port x Bit 10 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
9 (R/W)	PX9	Port x Bit 9 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
8 (R/W)	PX8	Port x Bit 8 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
7 (R/W)	PX7	Port x Bit 7 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
6 (R/W)	PX6	Port x Bit 6 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
5 (R/W)	PX5	Port x Bit 5 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
4 (R/W)	PX4	Port x Bit 4 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
3 (R/W)	PX3	Port x Bit 3 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
2 (R/W)	PX2	Port x Bit 2 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Table 12-14: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	PX1	Port x Bit 1 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.
0 (R/W)	PX0	Port x Bit 0 Direction.
		0 Input mode. The output driver is disabled.
		1 Output mode. The output driver is enabled.

Port x GPIO Direction Clear Register

The `PORT_DIR_CLR` register disables output mode and disables the output drivers for GPIO pins. For more information, see the `PORT_DIR` register description.

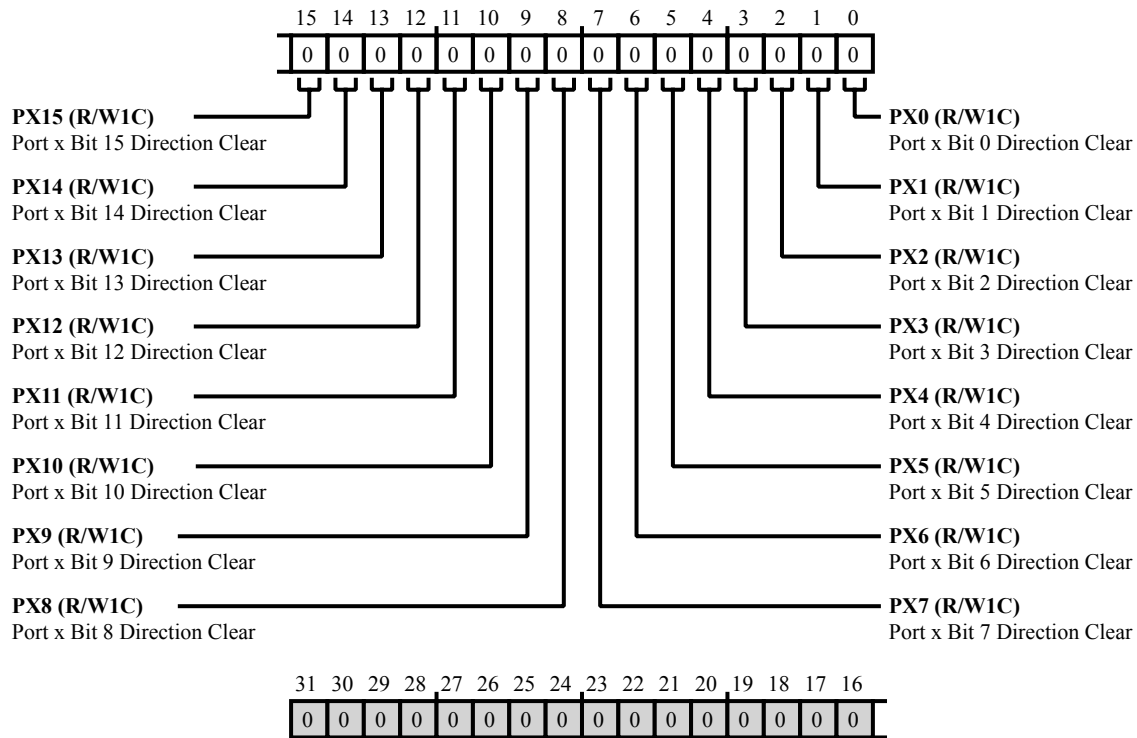


Figure 12-12: `PORT_DIR_CLR` Register Diagram

Table 12-15: `PORT_DIR_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PX15	Port x Bit 15 Direction Clear. The <code>PORT_DIR_CLR</code> . PX15 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
14 (R/W1C)	PX14	Port x Bit 14 Direction Clear. The <code>PORT_DIR_CLR</code> . PX14 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver

Table 12-15: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	PX13	Port x Bit 13 Direction Clear. The PORT_DIR_CLR.PX13 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
12 (R/W1C)	PX12	Port x Bit 12 Direction Clear. The PORT_DIR_CLR.PX12 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
11 (R/W1C)	PX11	Port x Bit 11 Direction Clear. The PORT_DIR_CLR.PX11 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
10 (R/W1C)	PX10	Port x Bit 10 Direction Clear. The PORT_DIR_CLR.PX10 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
9 (R/W1C)	PX9	Port x Bit 9 Direction Clear. The PORT_DIR_CLR.PX9 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
8 (R/W1C)	PX8	Port x Bit 8 Direction Clear. The PORT_DIR_CLR.PX8 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver

Table 12-15: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	PX7	Port x Bit 7 Direction Clear. The PORT_DIR_CLR.PX7 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
6 (R/W1C)	PX6	Port x Bit 6 Direction Clear. The PORT_DIR_CLR.PX6 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
5 (R/W1C)	PX5	Port x Bit 5 Direction Clear. The PORT_DIR_CLR.PX5 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
4 (R/W1C)	PX4	Port x Bit 4 Direction Clear. The PORT_DIR_CLR.PX4 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
3 (R/W1C)	PX3	Port x Bit 3 Direction Clear. The PORT_DIR_CLR.PX3 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
2 (R/W1C)	PX2	Port x Bit 2 Direction Clear. The PORT_DIR_CLR.PX2 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver

Table 12-15: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	PX1	Port x Bit 1 Direction Clear. The PORT_DIR_CLR.PX1 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver
0 (R/W1C)	PX0	Port x Bit 0 Direction Clear. The PORT_DIR_CLR.PX0 bit disables output mode and the output drivers for port x.
		0 No Effect
		1 Disable output mode/driver

Port x GPIO Direction Set Register

The `PORT_DIR_SET` register enables output mode and output drivers for GPIO pins. For more information, see the `PORT_DIR` register description.

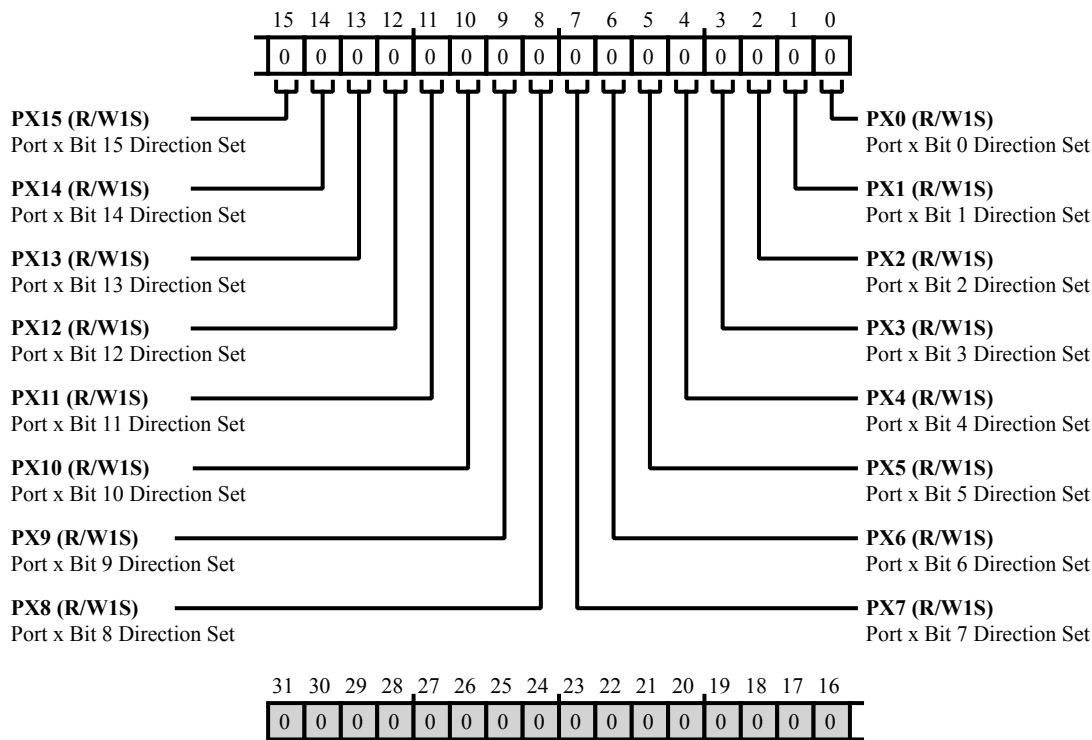


Figure 12-13: `PORT_DIR_SET` Register Diagram

Table 12-16: `PORT_DIR_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PX15	Port x Bit 15 Direction Set. The <code>PORT_DIR_SET.PX15</code> bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
14 (R/W1S)	PX14	Port x Bit 14 Direction Set. The <code>PORT_DIR_SET.PX14</code> bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver

Table 12-16: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1S)	PX13	Port x Bit 13 Direction Set. The PORT_DIR_SET.PX13 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
12 (R/W1S)	PX12	Port x Bit 12 Direction Set. The PORT_DIR_SET.PX12 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
11 (R/W1S)	PX11	Port x Bit 11 Direction Set. The PORT_DIR_SET.PX11 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
10 (R/W1S)	PX10	Port x Bit 10 Direction Set. The PORT_DIR_SET.PX10 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
9 (R/W1S)	PX9	Port x Bit 9 Direction Set. The PORT_DIR_SET.PX9 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
8 (R/W1S)	PX8	Port x Bit 8 Direction Set. The PORT_DIR_SET.PX8 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
7 (R/W1S)	PX7	Port x Bit 7 Direction Set. The PORT_DIR_SET.PX7 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver

Table 12-16: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1S)	PX6	Port x Bit 6 Direction Set. The PORT_DIR_SET.PX6 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
5 (R/W1S)	PX5	Port x Bit 5 Direction Set. The PORT_DIR_SET.PX5 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
4 (R/W1S)	PX4	Port x Bit 4 Direction Set. The PORT_DIR_SET.PX4 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
3 (R/W1S)	PX3	Port x Bit 3 Direction Set. The PORT_DIR_SET.PX3 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
2 (R/W1S)	PX2	Port x Bit 2 Direction Set. The PORT_DIR_SET.PX2 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
1 (R/W1S)	PX1	Port x Bit 1 Direction Set. The PORT_DIR_SET.PX1 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver
0 (R/W1S)	PX0	Port x Bit 0 Direction Set. The PORT_DIR_SET.PX0 bit enables the output mode/driver for port x.
		0 No Effect
		1 Enable output mode/driver

Port x Function Enable Register

The `PORT_FER` register bits indicate each port bit's operating mode: general purpose I/O mode or peripheral mode. After reset, all pins default to GPIO mode. Setting a bit in the `PORT_FER` registers enables a peripheral module to take ownership of the pin. The function enable bits impact output control only. Regardless of the setting of the function enable bits, both GPIO and peripherals can still sense the pin input. After a function is enabled, it is up to the `PORT_MUX` registers as to which peripheral takes control.

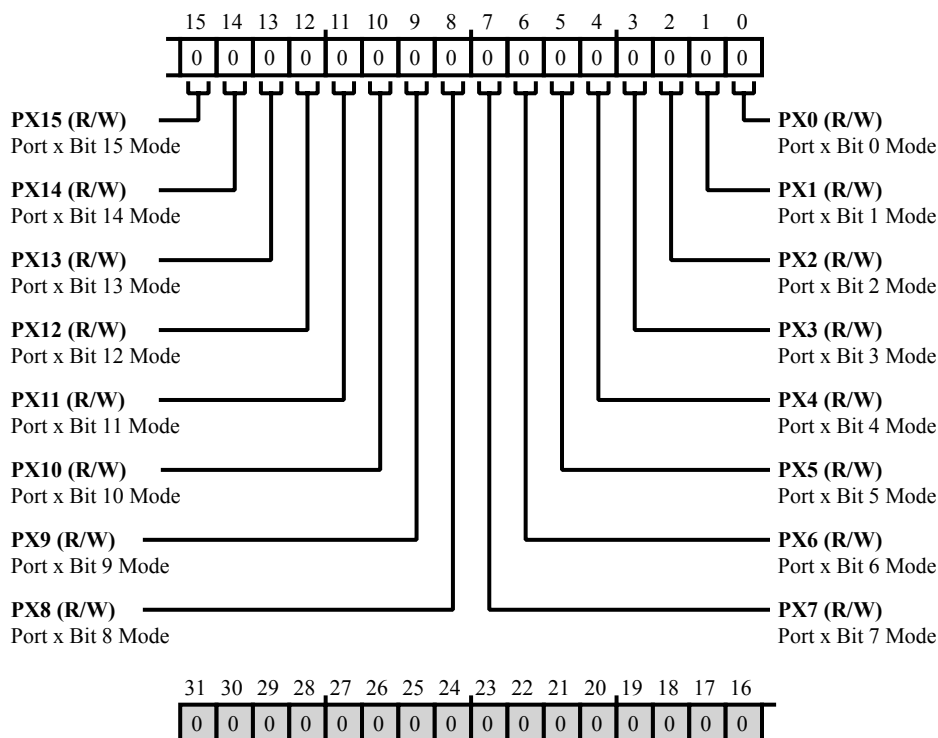


Figure 12-14: PORT_FER Register Diagram

Table 12-17: PORT_FER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	Port x Bit 15 Mode. The <code>PORT_FER.PX15</code> bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
14 (R/W)	PX14	Port x Bit 14 Mode. The <code>PORT_FER.PX14</code> bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode

Table 12-17: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	PX13	Port x Bit 13 Mode. The PORT_FER.PX13 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
12 (R/W)	PX12	Port x Bit 12 Mode. The PORT_FER.PX12 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
11 (R/W)	PX11	Port x Bit 11 Mode. The PORT_FER.PX11 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
10 (R/W)	PX10	Port x Bit 10 Mode. The PORT_FER.PX10 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
9 (R/W)	PX9	Port x Bit 9 Mode. The PORT_FER.PX9 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
8 (R/W)	PX8	Port x Bit 8 Mode. The PORT_FER.PX8 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
7 (R/W)	PX7	Port x Bit 7 Mode. The PORT_FER.PX7 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode

Table 12-17: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	PX6	Port x Bit 6 Mode. The PORT_FER.PX6 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
5 (R/W)	PX5	Port x Bit 5 Mode. The PORT_FER.PX5 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
4 (R/W)	PX4	Port x Bit 4 Mode. The PORT_FER.PX4 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
3 (R/W)	PX3	Port x Bit 3 Mode. The PORT_FER.PX3 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
2 (R/W)	PX2	Port x Bit 2 Mode. The PORT_FER.PX2 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
1 (R/W)	PX1	Port x Bit 1 Mode. The PORT_FER.PX1 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode
0 (R/W)	PX0	Port x Bit 0 Mode. The PORT_FER.PX0 bit indicates the operating mode for port x.
		0 GPIO Mode
		1 Peripheral Mode

Port x Function Enable Clear Register

The `PORT_FER_CLR` register permits enabling GPIO mode for each bit and corresponding GPIO pin. Writing 1 to a bit in `PORT_FER_CLR` enables GPIO mode for the corresponding pin.

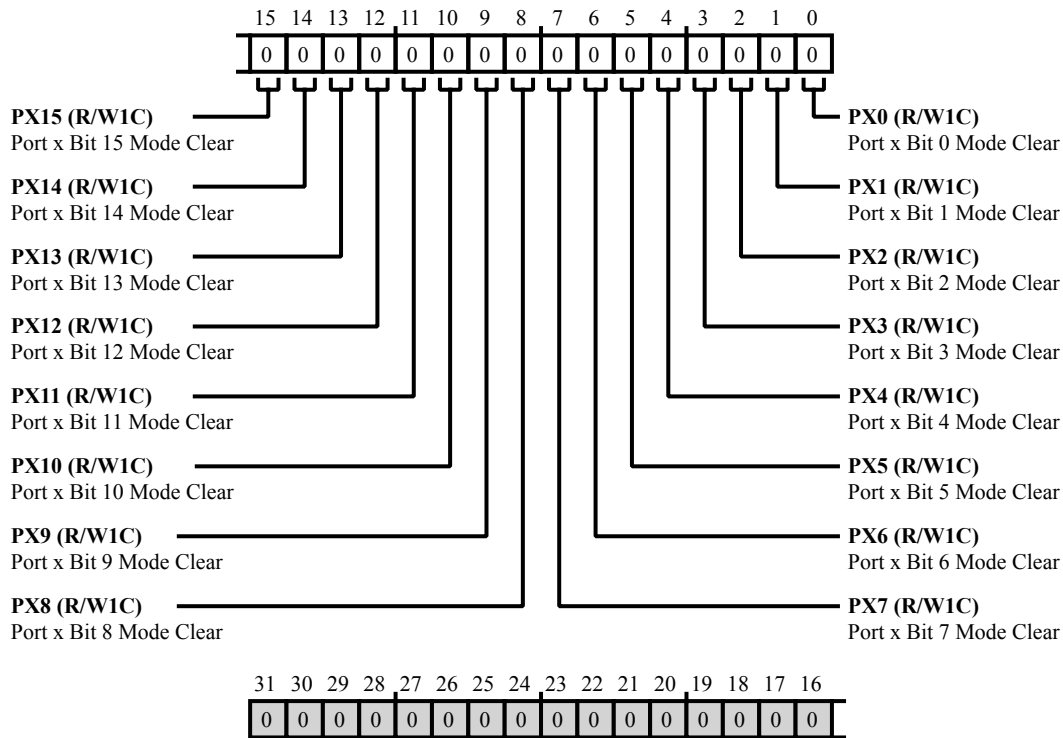


Figure 12-15: `PORT_FER_CLR` Register Diagram

Table 12-18: `PORT_FER_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PX15	Port x Bit 15 Mode Clear. The <code>PORT_FER_CLR</code> . PX15 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
14 (R/W1C)	PX14	Port x Bit 14 Mode Clear. The <code>PORT_FER_CLR</code> . PX14 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode

Table 12-18: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	PX13	Port x Bit 13 Mode Clear. The PORT_FER_CLR.PX13 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
12 (R/W1C)	PX12	Port x Bit 12 Mode Clear. The PORT_FER_CLR.PX12 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
11 (R/W1C)	PX11	Port x Bit 11 Mode Clear. The PORT_FER_CLR.PX11 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
10 (R/W1C)	PX10	Port x Bit 10 Mode Clear. The PORT_FER_CLR.PX10 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
9 (R/W1C)	PX9	Port x Bit 9 Mode Clear. The PORT_FER_CLR.PX9 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
8 (R/W1C)	PX8	Port x Bit 8 Mode Clear. The PORT_FER_CLR.PX8 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
7 (R/W1C)	PX7	Port x Bit 7 Mode Clear. The PORT_FER_CLR.PX7 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode

Table 12-18: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	PX6	Port x Bit 6 Mode Clear. The PORT_FER_CLR.PX6 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
5 (R/W1C)	PX5	Port x Bit 5 Mode Clear. The PORT_FER_CLR.PX5 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
4 (R/W1C)	PX4	Port x Bit 4 Mode Clear. The PORT_FER_CLR.PX4 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
3 (R/W1C)	PX3	Port x Bit 3 Mode Clear. The PORT_FER_CLR.PX3 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
2 (R/W1C)	PX2	Port x Bit 2 Mode Clear. The PORT_FER_CLR.PX2 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
1 (R/W1C)	PX1	Port x Bit 1 Mode Clear. The PORT_FER_CLR.PX1 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode
0 (R/W1C)	PX0	Port x Bit 0 Mode Clear. The PORT_FER_CLR.PX0 bit enables GPIO mode.
		0 No Effect
		1 Set Bit for GPIO Mode

Port x Function Enable Set Register

The `PORT_FER_SET` register permits enabling peripheral mode for each bit and corresponding GPIO pin. Writing 1 to a bit in `PORT_FER_SET` enables peripheral mode for the corresponding pin.

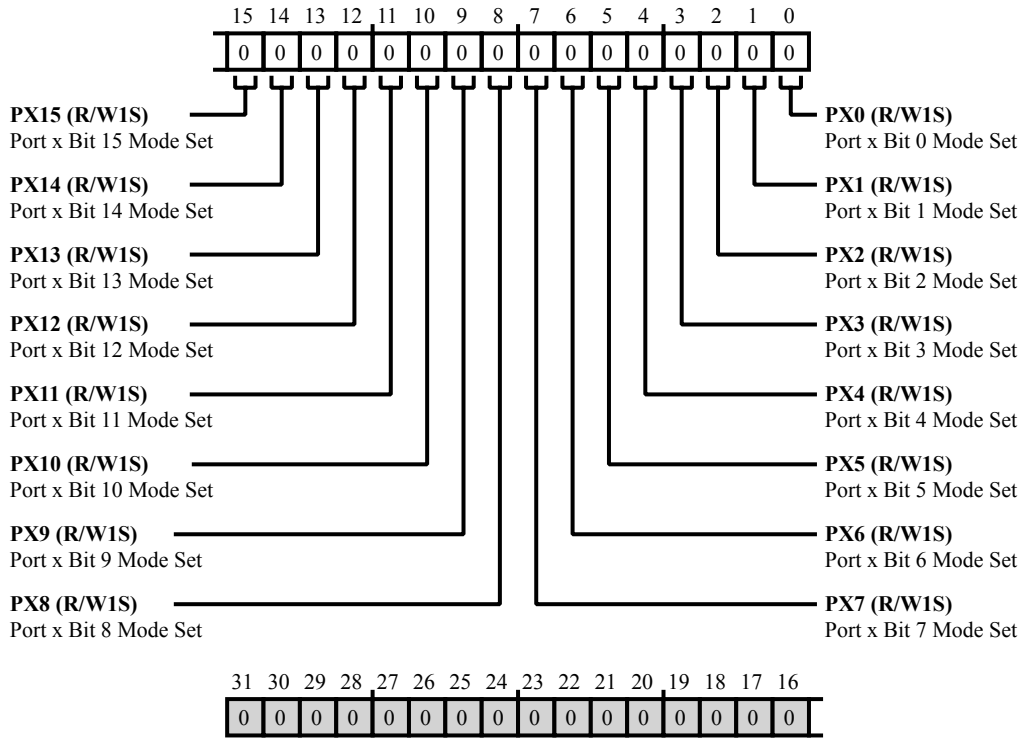


Figure 12-16: `PORT_FER_SET` Register Diagram

Table 12-19: `PORT_FER_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PX15	Port x Bit 15 Mode Set. The <code>PORT_FER_SET.PX15</code> bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
14 (R/W1S)	PX14	Port x Bit 14 Mode Set. The <code>PORT_FER_SET.PX14</code> bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode

Table 12-19: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1S)	PX13	Port x Bit 13 Mode Set. The PORT_FER_SET.PX13 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
12 (R/W1S)	PX12	Port x Bit 12 Mode Set. The PORT_FER_SET.PX12 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
11 (R/W1S)	PX11	Port x Bit 11 Mode Set. The PORT_FER_SET.PX11 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
10 (R/W1S)	PX10	Port x Bit 10 Mode Set. The PORT_FER_SET.PX10 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
9 (R/W1S)	PX9	Port x Bit 9 Mode Set. The PORT_FER_SET.PX9 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
8 (R/W1S)	PX8	Port x Bit 8 Mode Set. The PORT_FER_SET.PX8 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
7 (R/W1S)	PX7	Port x Bit 7 Mode Set. The PORT_FER_SET.PX7 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode

Table 12-19: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1S)	PX6	Port x Bit 6 Mode Set. The PORT_FER_SET.PX6 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
5 (R/W1S)	PX5	Port x Bit 5 Mode Set. The PORT_FER_SET.PX5 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
4 (R/W1S)	PX4	Port x Bit 4 Mode Set. The PORT_FER_SET.PX4 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
3 (R/W1S)	PX3	Port x Bit 3 Mode Set. The PORT_FER_SET.PX3 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
2 (R/W1S)	PX2	Port x Bit 2 Mode Set. The PORT_FER_SET.PX2 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
1 (R/W1S)	PX1	Port x Bit 1 Mode Set. The PORT_FER_SET.PX1 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode
0 (R/W1S)	PX0	Port x Bit 0 Mode Set. The PORT_FER_SET.PX0 bit enables peripheral mode.
		0 No Effect
		1 Set Bit for Peripheral Mode

Port x GPIO Input Enable Register

The `PORT_INEN`, `PORT_INEN_SET`, and `PORT_INEN_CLR` registers enable or disable input drivers, which are required for using a GPIO pin in input mode.

Writes to the `PORT_INEN` register affect the input drivers for all pins of the port. To set or clear specific pin drivers without impacting other pin drivers of the port, use the `PORT_INEN_SET` and `PORT_INEN_CLR` registers.

If the input is enabled, reads from the `PORT_DATA`, `PORT_DATA_SET`, or `PORT_DATA_CLR` registers return the state of the pins. However, the state of the output is not overwritten by the input. It is altered by software writes only. Input and output drivers can be enabled at the same time. In this case, a read of the data register returns the true value of the data register and not the pin state.

For more information, see the `PORT_DATA` register description and the `PORT_DIR` register description.

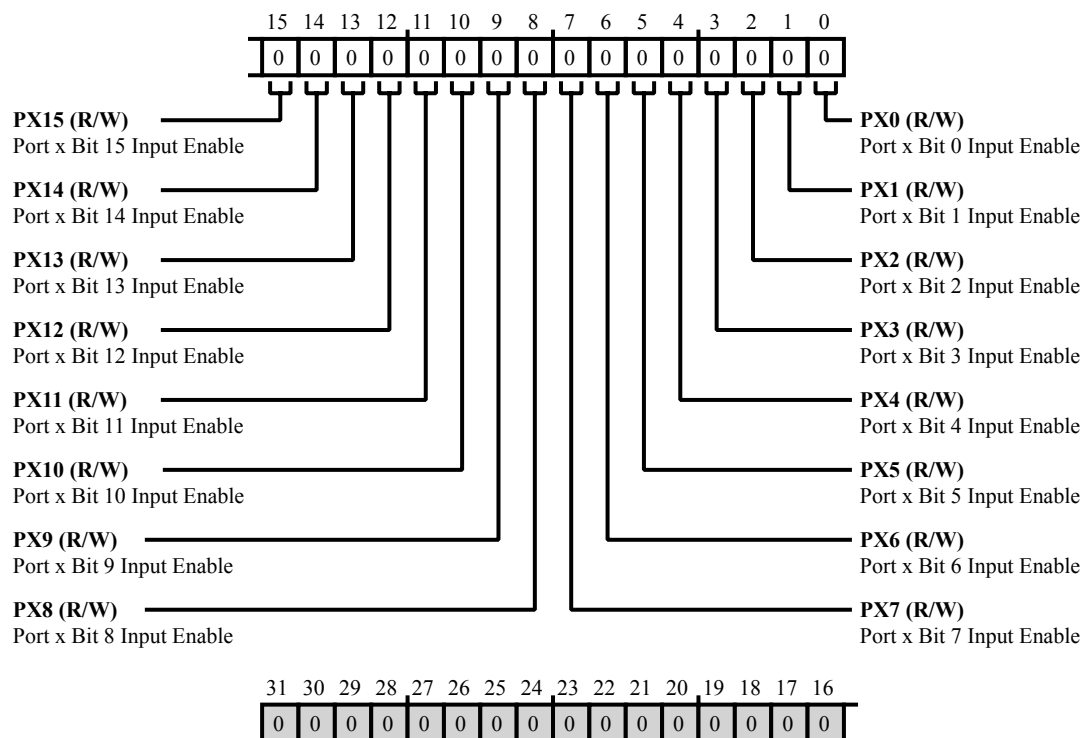


Figure 12-17: PORT_INEN Register Diagram

Table 12-20: PORT_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 12-20: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	PX14	Port x Bit 14 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
13 (R/W)	PX13	Port x Bit 13 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
12 (R/W)	PX12	Port x Bit 12 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
11 (R/W)	PX11	Port x Bit 11 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
10 (R/W)	PX10	Port x Bit 10 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
9 (R/W)	PX9	Port x Bit 9 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
8 (R/W)	PX8	Port x Bit 8 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
7 (R/W)	PX7	Port x Bit 7 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
6 (R/W)	PX6	Port x Bit 6 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
5 (R/W)	PX5	Port x Bit 5 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver

Table 12-20: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	PX4	Port x Bit 4 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
3 (R/W)	PX3	Port x Bit 3 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
2 (R/W)	PX2	Port x Bit 2 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
1 (R/W)	PX1	Port x Bit 1 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver
0 (R/W)	PX0	Port x Bit 0 Input Enable.
		0 Disable Input Driver
		1 Enable Input Driver

Port x GPIO Input Enable Clear Register

The `PORT_INEN_CLR` register disables the input drivers for GPIO pins. For more information, see the `PORT_INEN` register description.

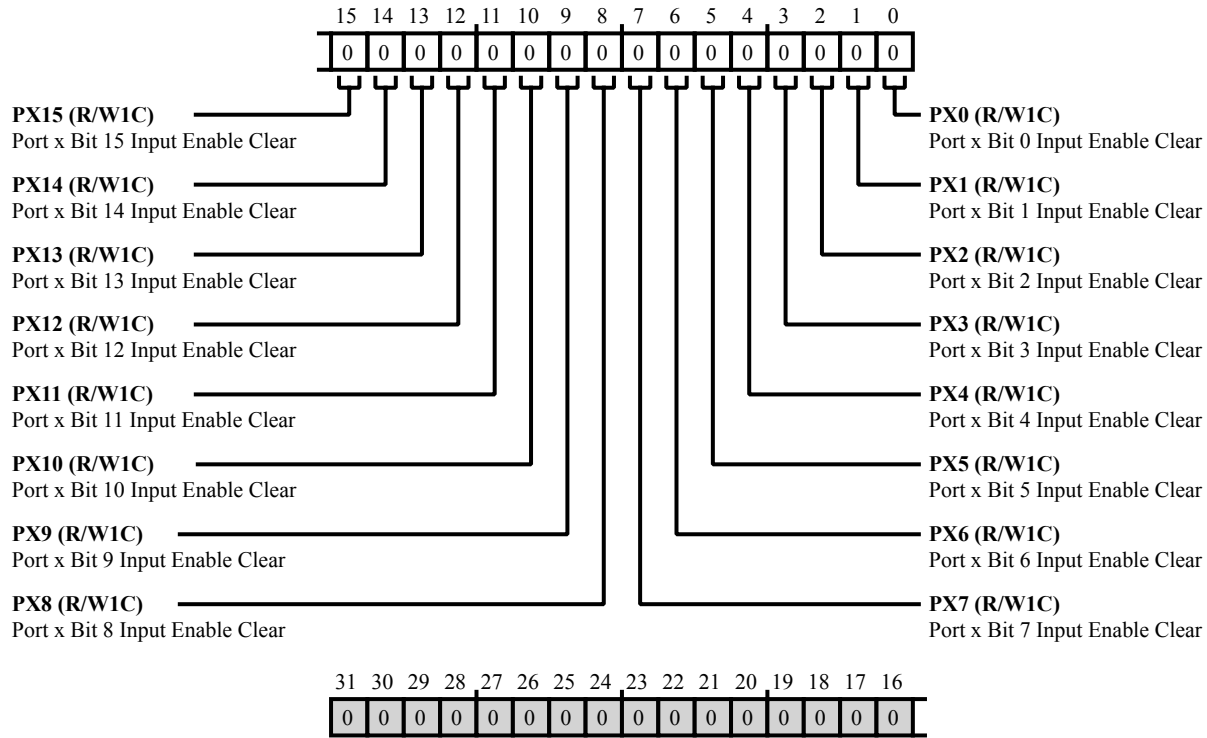


Figure 12-18: `PORT_INEN_CLR` Register Diagram

Table 12-21: `PORT_INEN_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PX15	Port x Bit 15 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
14 (R/W1C)	PX14	Port x Bit 14 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
13 (R/W1C)	PX13	Port x Bit 13 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.

Table 12-21: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W1C)	PX12	Port x Bit 12 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
11 (R/W1C)	PX11	Port x Bit 11 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
10 (R/W1C)	PX10	Port x Bit 10 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
9 (R/W1C)	PX9	Port x Bit 9 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
8 (R/W1C)	PX8	Port x Bit 8 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
7 (R/W1C)	PX7	Port x Bit 7 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
6 (R/W1C)	PX6	Port x Bit 6 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
5 (R/W1C)	PX5	Port x Bit 5 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
4 (R/W1C)	PX4	Port x Bit 4 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
3 (R/W1C)	PX3	Port x Bit 3 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.

Table 12-21: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	PX2	Port x Bit 2 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
1 (R/W1C)	PX1	Port x Bit 1 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.
0 (R/W1C)	PX0	Port x Bit 0 Input Enable Clear.
		0 No Effect
		1 Clear Bit. Set to disable the input driver.

Port x GPIO Input Enable Set Register

The `PORT_INEN_SET` register enables input drivers for GPIO pins. For more information, see the `PORT_INEN` register description.

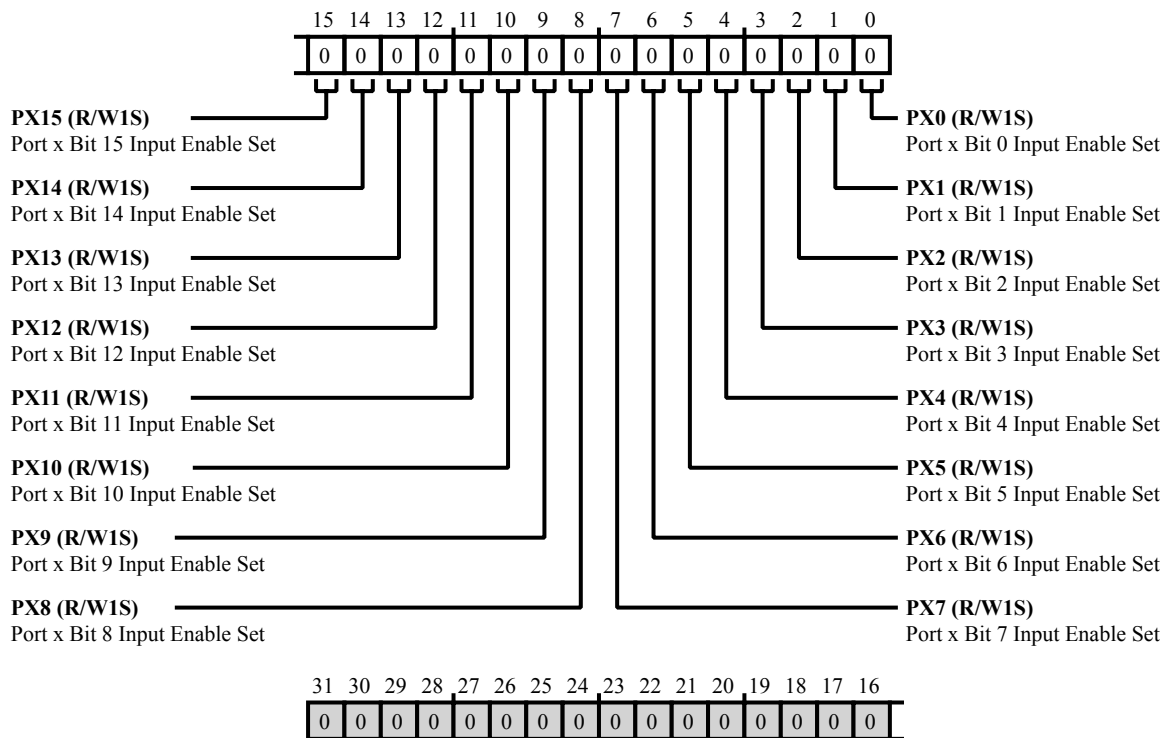


Figure 12-19: `PORT_INEN_SET` Register Diagram

Table 12-22: `PORT_INEN_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PX15	Port x Bit 15 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
14 (R/W1S)	PX14	Port x Bit 14 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
13 (R/W1S)	PX13	Port x Bit 13 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.

Table 12-22: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W1S)	PX12	Port x Bit 12 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
11 (R/W1S)	PX11	Port x Bit 11 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
10 (R/W1S)	PX10	Port x Bit 10 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
9 (R/W1S)	PX9	Port x Bit 9 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
8 (R/W1S)	PX8	Port x Bit 8 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
7 (R/W1S)	PX7	Port x Bit 7 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
6 (R/W1S)	PX6	Port x Bit 6 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
5 (R/W1S)	PX5	Port x Bit 5 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
4 (R/W1S)	PX4	Port x Bit 4 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
3 (R/W1S)	PX3	Port x Bit 3 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.

Table 12-22: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1S)	PX2	Port x Bit 2 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
1 (R/W1S)	PX1	Port x Bit 1 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.
0 (R/W1S)	PX0	Port x Bit 0 Input Enable Set.
		0 No Effect
		1 Set Bit. Set to enable the input driver.

Port x GPIO Lock Register

The `PORT_LOCK` register enables (unlocks) or disables (locks) write access selectively for the PORT control registers.

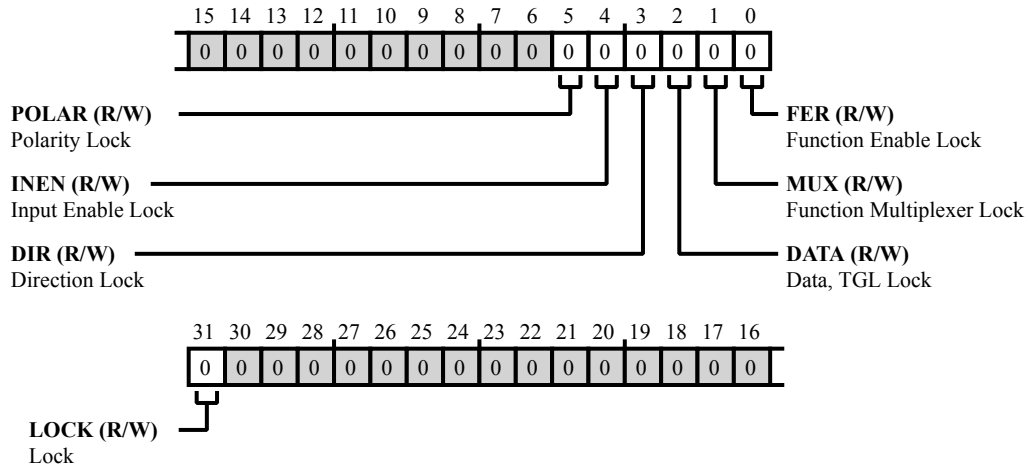


Figure 12-20: PORT_LOCK Register Diagram

Table 12-23: PORT_LOCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set (<code>SPU_CTL.GLCK</code> bit =1) and the <code>PORT_LOCK.LOCK</code> bit is set, the <code>PORT_LOCK</code> register is read only (locked).
		0 Unlock
		1 Lock
5 (R/W)	POLAR	Polarity Lock. The <code>PORT_LOCK.POLAR</code> disables write access to the <code>PORT_POL</code> , <code>PORT_POL_SET</code> , and <code>PORT_POL_CLR</code> registers.
		0 Unlock POL
		1 Lock POL
4 (R/W)	INEN	Input Enable Lock. The <code>PORT_LOCK.INEN</code> disables write access to the <code>PORT_INEN</code> , <code>PORT_INEN_SET</code> , and <code>PORT_INEN_CLR</code> registers.
		0 Unlock INEN
		1 Lock INEN

Table 12-23: PORT_LOCK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	DIR	Direction Lock. The PORT_LOCK.DIR disables write access to the PORT_DIR, PORT_DIR_SET, PORT_DIR_CLR registers.
		0 Unlock DIR
		1 Lock DIR
2 (R/W)	DATA	Data, TGL Lock. The PORT_LOCK.DATA disables write access to the PORT_DATA, PORT_DATA_SET, PORT_DATA_CLR, and PORT_DATA_TGL registers.
		0 Unlock DATA
		1 Lock DATA
1 (R/W)	MUX	Function Multiplexer Lock. The PORT_LOCK.MUX disables write accesses to the PORT_MUX register.
		0 Unlock MUX
		1 Lock MUX
0 (R/W)	FER	Function Enable Lock. The PORT_LOCK.FER disables write access to the PORT_FER, PORT_FER_SET, and PORT_FER_CLR registers.
		0 Unlock FER
		1 Lock FER

Port x Multiplexer Control Register

When a pin is in peripheral mode (not GPIO mode), the `PORT_MUX` register controls which peripheral takes ownership of a pin. Ports may have multiple, different peripheral functions. Two bits are required to describe every multiplexer on an individual pin-by-pin scheme. For example, bit 0 and bit 1 of the `PORT_MUX` register control the multiplexer of pin 0, bit 2 and bit 3 of `PORT_MUX` control the multiplexer of pin 1, and so on. The value of any `PORT_MUX` bit has no effect on the port pins when the associated bit in the `PORT_FER` register is 0 (selects GPIO mode). Even if a port has only one function, the `PORT_MUX` register is still present. For single function ports (no multiplexing is needed), leave the `PORT_MUX` bits at 0 (default). For all `PORT_MUX` bit fields: 00 = default/reset peripheral option, 01 = first alternate peripheral option, 10 = second alternate peripheral option, and 11 = third alternate peripheral option.

See the processor data sheet for details regarding the peripheral options associated with each port.

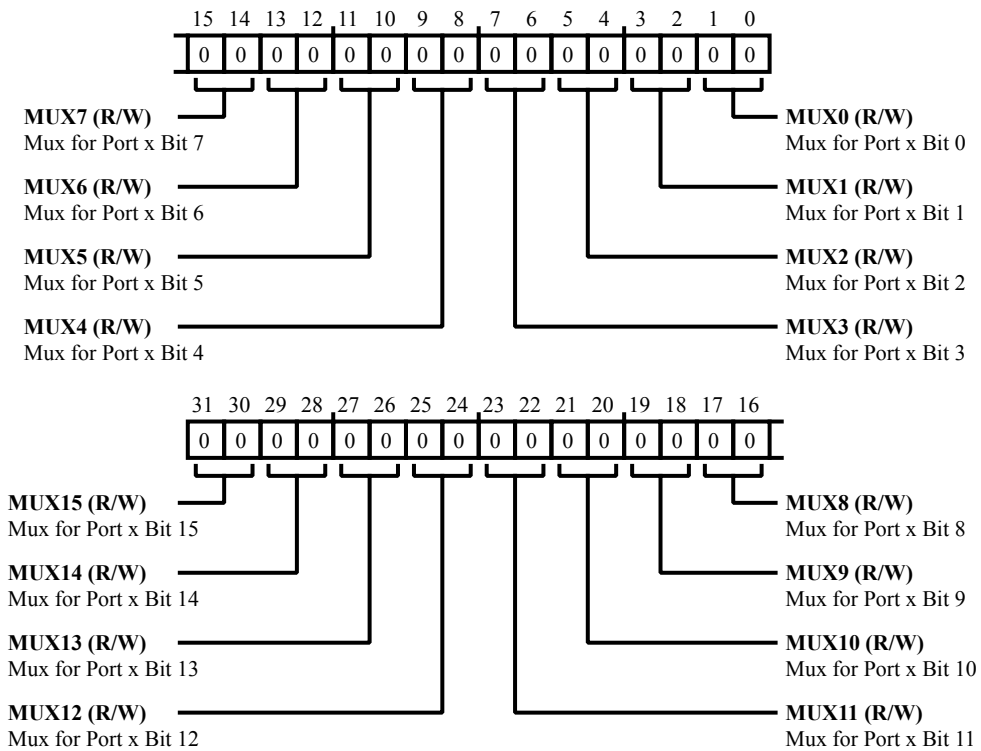


Figure 12-21: PORT_MUX Register Diagram

Table 12-24: PORT_MUX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MUX15	Mux for Port x Bit 15. The <code>PORT_MUX.MUX15</code> bit provides multiplexer control for port x bit 15.

Table 12-24: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:28 (R/W)	MUX14	Mux for Port x Bit 14. The PORT_MUX.MUX14 bit provides multiplexer control for port x bit 14.
27:26 (R/W)	MUX13	Mux for Port x Bit 13. The PORT_MUX.MUX13 bit provides multiplexer control for port x bit 13.
25:24 (R/W)	MUX12	Mux for Port x Bit 12. The PORT_MUX.MUX12 bit provides multiplexer control for port x bit 12.
23:22 (R/W)	MUX11	Mux for Port x Bit 11. The PORT_MUX.MUX11 bit provides multiplexer control for port x bit 11.
21:20 (R/W)	MUX10	Mux for Port x Bit 10. The PORT_MUX.MUX10 bit provides multiplexer control for port x bit 10.
19:18 (R/W)	MUX9	Mux for Port x Bit 9. The PORT_MUX.MUX9 bit provides multiplexer control for port x bit 9.
17:16 (R/W)	MUX8	Mux for Port x Bit 8. The PORT_MUX.MUX8 bit provides multiplexer control for port x bit 8.
15:14 (R/W)	MUX7	Mux for Port x Bit 7. The PORT_MUX.MUX7 bit provides multiplexer control for port x bit 7.
13:12 (R/W)	MUX6	Mux for Port x Bit 6. The PORT_MUX.MUX6 bit provides multiplexer control for port x bit 6.
11:10 (R/W)	MUX5	Mux for Port x Bit 5. The PORT_MUX.MUX5 bit provides multiplexer control for port x bit 5.
9:8 (R/W)	MUX4	Mux for Port x Bit 4. The PORT_MUX.MUX4 bit provides multiplexer control for port x bit 4.
7:6 (R/W)	MUX3	Mux for Port x Bit 3. The PORT_MUX.MUX3 bit provides multiplexer control for port x bit 3.
5:4 (R/W)	MUX2	Mux for Port x Bit 2. The PORT_MUX.MUX2 bit provides multiplexer control for port x bit 2.
3:2 (R/W)	MUX1	Mux for Port x Bit 1. The PORT_MUX.MUX1 bit provides multiplexer control for port x bit 1.
1:0 (R/W)	MUX0	Mux for Port x Bit 0. The PORT_MUX.MUX0 bit provides multiplexer control for port x bit 0.

Port x GPIO Polarity Invert Register

The `PORT_POL`, `PORT_POL_SET`, and `PORT_POL_CLR` registers enable or disable inverting polarity of GPIO signals. To invert polarity of peripheral signals, use the inversion selection programming in the signal's corresponding module.

Writes to the `PORT_POL` register affect the polarity inversion selection of all pins of the port. To enable or disable polarity inversion for specific pins without impacting other pins of the port, use the `PORT_POL_SET` and `PORT_POL_CLR` registers.

Setting a bit in the `PORT_POL` register enables polarity inversion on the corresponding inversion GPIO pin, making the pin active-low or falling-edge sensitive. Clearing a bit in the `PORT_POL` register disables polarity (default state) on the corresponding GPIO pin, making it active-high or rising-edge sensitive.

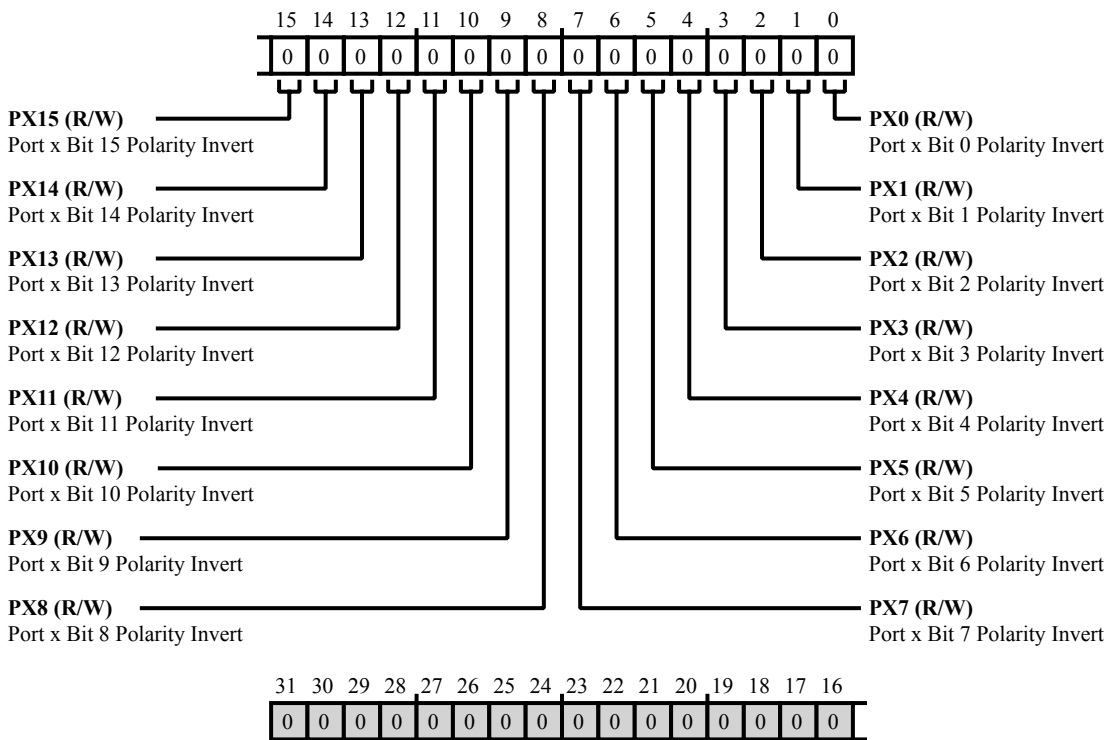


Figure 12-22: PORT_POL Register Diagram

Table 12-25: PORT_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	Port x Bit 15 Polarity Invert. The <code>PORT_POL.PX15</code> bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.

Table 12-25: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	PX14	Port x Bit 14 Polarity Invert. The PORT_POL.PX14 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
13 (R/W)	PX13	Port x Bit 13 Polarity Invert. The PORT_POL.PX13 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
12 (R/W)	PX12	Port x Bit 12 Polarity Invert. The PORT_POL.PX12 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
11 (R/W)	PX11	Port x Bit 11 Polarity Invert. The PORT_POL.PX11 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
10 (R/W)	PX10	Port x Bit 10 Polarity Invert. The PORT_POL.PX10 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
9 (R/W)	PX9	Port x Bit 9 Polarity Invert. The PORT_POL.PX9 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
8 (R/W)	PX8	Port x Bit 8 Polarity Invert. The PORT_POL.PX8 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.

Table 12-25: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	PX7	Port x Bit 7 Polarity Invert. The PORT_POL.PX7 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
6 (R/W)	PX6	Port x Bit 6 Polarity Invert. The PORT_POL.PX6 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
5 (R/W)	PX5	Port x Bit 5 Polarity Invert. The PORT_POL.PX5 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
4 (R/W)	PX4	Port x Bit 4 Polarity Invert. The PORT_POL.PX4 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
3 (R/W)	PX3	Port x Bit 3 Polarity Invert. The PORT_POL.PX3 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
2 (R/W)	PX2	Port x Bit 2 Polarity Invert. The PORT_POL.PX2 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.
1 (R/W)	PX1	Port x Bit 1 Polarity Invert. The PORT_POL.PX1 bit enables polarity inversion.
		0 No Invert. GPIO is active high or rising edge sensitive.
		1 Invert. GPIO is active low or falling edge sensitive.

Table 12-25: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PX0	Port x Bit 0 Polarity Invert. The PORT_POL.PX0 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Port x GPIO Polarity Invert Clear Register

The `PORT_POL_CLR` register disables polarity inversion for GPIO pins. For more information, see the `PORT_POL` register description.

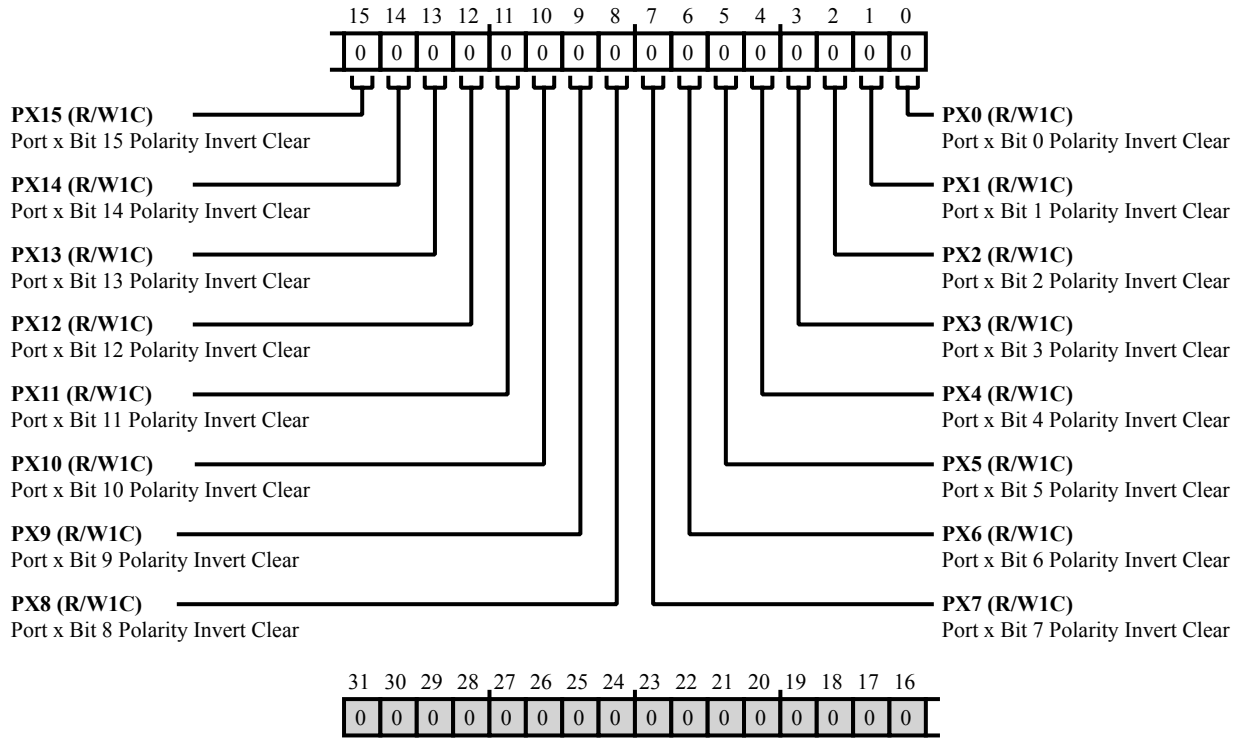


Figure 12-23: `PORT_POL_CLR` Register Diagram

Table 12-26: `PORT_POL_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PX15	Port x Bit 15 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
14 (R/W1C)	PX14	Port x Bit 14 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
13 (R/W1C)	PX13	Port x Bit 13 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.

Table 12-26: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W1C)	PX12	Port x Bit 12 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
11 (R/W1C)	PX11	Port x Bit 11 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
10 (R/W1C)	PX10	Port x Bit 10 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
9 (R/W1C)	PX9	Port x Bit 9 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
8 (R/W1C)	PX8	Port x Bit 8 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
7 (R/W1C)	PX7	Port x Bit 7 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
6 (R/W1C)	PX6	Port x Bit 6 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
5 (R/W1C)	PX5	Port x Bit 5 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
4 (R/W1C)	PX4	Port x Bit 4 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
3 (R/W1C)	PX3	Port x Bit 3 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.

Table 12-26: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	PX2	Port x Bit 2 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
1 (R/W1C)	PX1	Port x Bit 1 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.
0 (R/W1C)	PX0	Port x Bit 0 Polarity Invert Clear.
		0 No Effect
		1 Clear Bit. Set to disable GPIO pin polarity invert.

Port x GPIO Polarity Invert Set Register

The `PORT_POL_SET` register enables polarity inversion for GPIO pins. For more information, see the `PORT_POL` register description.

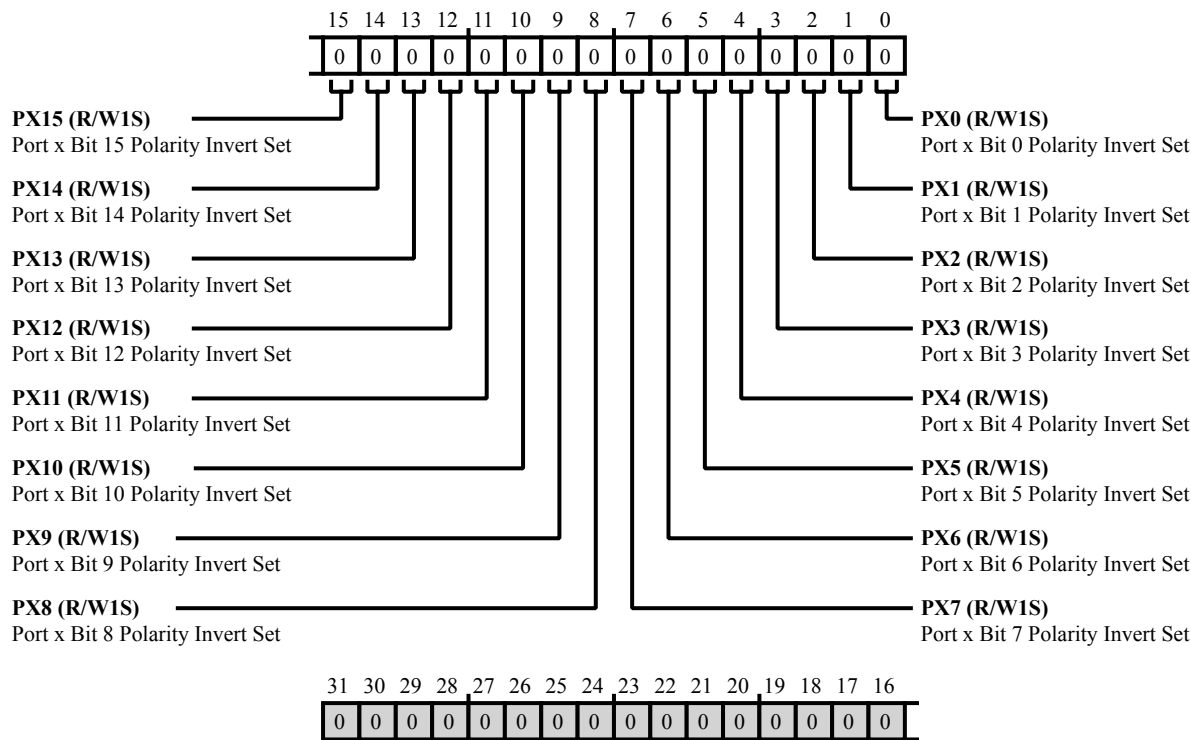


Figure 12-24: `PORT_POL_SET` Register Diagram

Table 12-27: `PORT_POL_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PX15	Port x Bit 15 Polarity Invert Set. The <code>PORT_POL_SET.PX15</code> bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
14 (R/W1S)	PX14	Port x Bit 14 Polarity Invert Set. The <code>PORT_POL_SET.PX14</code> bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.

Table 12-27: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1S)	PX13	Port x Bit 13 Polarity Invert Set. The PORT_POL_SET.PX13 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
12 (R/W1S)	PX12	Port x Bit 12 Polarity Invert Set. The PORT_POL_SET.PX12 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
11 (R/W1S)	PX11	Port x Bit 11 Polarity Invert Set. The PORT_POL_SET.PX11 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
10 (R/W1S)	PX10	Port x Bit 10 Polarity Invert Set. The PORT_POL_SET.PX10 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
9 (R/W1S)	PX9	Port x Bit 9 Polarity Invert Set. The PORT_POL_SET.PX9 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
8 (R/W1S)	PX8	Port x Bit 8 Polarity Invert Set. The PORT_POL_SET.PX8 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
7 (R/W1S)	PX7	Port x Bit 7 Polarity Invert Set. The PORT_POL_SET.PX7 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.

Table 12-27: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1S)	PX6	Port x Bit 6 Polarity Invert Set. The PORT_POL_SET . PX6 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
5 (R/W1S)	PX5	Port x Bit 5 Polarity Invert Set. The PORT_POL_SET . PX5 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
4 (R/W1S)	PX4	Port x Bit 4 Polarity Invert Set. The PORT_POL_SET . PX4 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
3 (R/W1S)	PX3	Port x Bit 3 Polarity Invert Set. The PORT_POL_SET . PX3 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
2 (R/W1S)	PX2	Port x Bit 2 Polarity Invert Set. The PORT_POL_SET . PX2 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
1 (R/W1S)	PX1	Port x Bit 1 Polarity Invert Set. The PORT_POL_SET . PX1 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.
0 (R/W1S)	PX0	Port x Bit 0 Polarity Invert Set. The PORT_POL_SET . PX0 bit enables pin polarity inversion.
		0 No Effect
		1 Set Bit. Set to enable GPIO pin polarity invert.

Port x GPIO Trigger Toggle Register

The `PORT_TRIG_TGL` register permits toggling the state of output GPIO pins in response to a trigger from the TRU for the corresponding port. Setting bits in the `PORT_TRIG_TGL` register enables triggers to toggle the state of those specific pins without impacting other pins of the port.

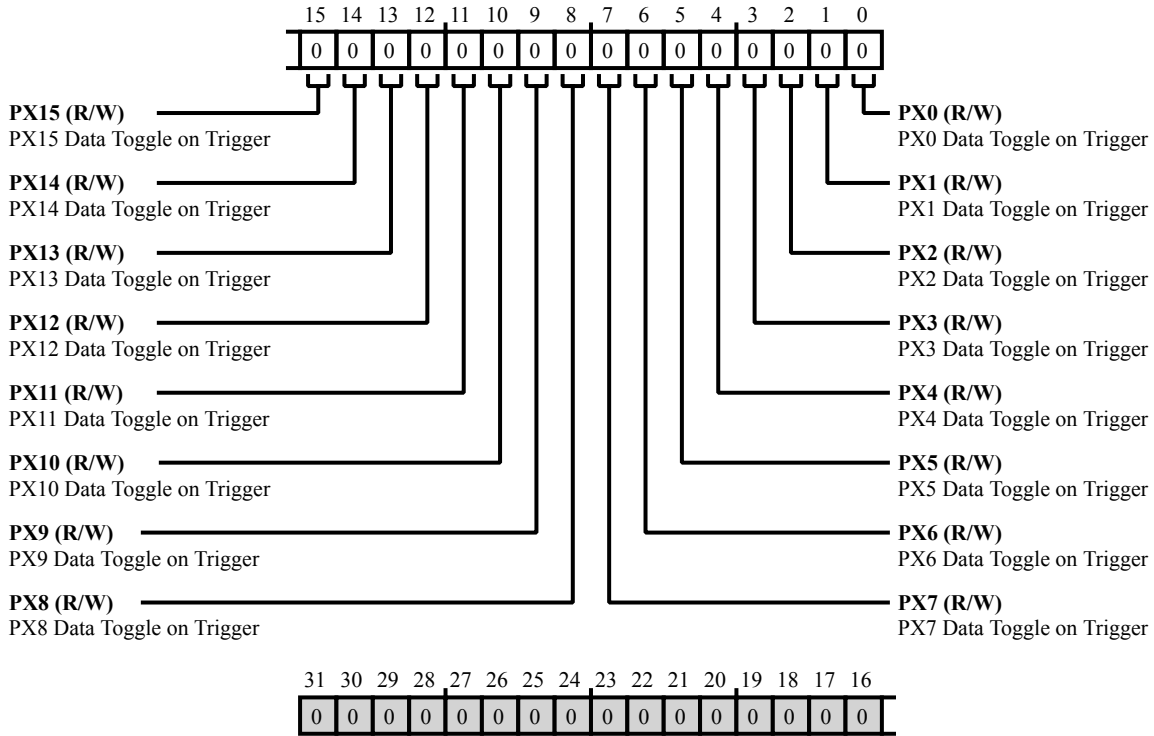


Figure 12-25: PORT_TRIG_TGL Register Diagram

Table 12-28: PORT_TRIG_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	PX15 Data Toggle on Trigger. The <code>PORT_TRIG_TGL.PX15</code> bit enables triggers to toggle the state of the pin.
14 (R/W)	PX14	PX14 Data Toggle on Trigger. The <code>PORT_TRIG_TGL.PX14</code> bit enables triggers to toggle the state of the pin.
13 (R/W)	PX13	PX13 Data Toggle on Trigger. The <code>PORT_TRIG_TGL.PX13</code> bit enables triggers to toggle the state of the pin.
12 (R/W)	PX12	PX12 Data Toggle on Trigger. The <code>PORT_TRIG_TGL.PX12</code> bit enables triggers to toggle the state of the pin.
11 (R/W)	PX11	PX11 Data Toggle on Trigger. The <code>PORT_TRIG_TGL.PX11</code> bit enables triggers to toggle the state of the pin.

Table 12-28: PORT_TRIG_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	PX10	PX10 Data Toggle on Trigger. The PORT_TRIG_TGL.PX10 bit enables triggers to toggle the state of the pin.
9 (R/W)	PX9	PX9 Data Toggle on Trigger. The PORT_TRIG_TGL.PX9 bit enables triggers to toggle the state of the pin.
8 (R/W)	PX8	PX8 Data Toggle on Trigger. The PORT_TRIG_TGL.PX8 bit enables triggers to toggle the state of the pin.
7 (R/W)	PX7	PX7 Data Toggle on Trigger. The PORT_TRIG_TGL.PX7 bit enables triggers to toggle the state of the pin.
6 (R/W)	PX6	PX6 Data Toggle on Trigger. The PORT_TRIG_TGL.PX6 bit enables triggers to toggle the state of the pin.
5 (R/W)	PX5	PX5 Data Toggle on Trigger. The PORT_TRIG_TGL.PX5 bit enables triggers to toggle the state of the pin.
4 (R/W)	PX4	PX4 Data Toggle on Trigger. The PORT_TRIG_TGL.PX4 bit enables triggers to toggle the state of the pin.
3 (R/W)	PX3	PX3 Data Toggle on Trigger. The PORT_TRIG_TGL.PX3 bit enables triggers to toggle the state of the pin.
2 (R/W)	PX2	PX2 Data Toggle on Trigger. The PORT_TRIG_TGL.PX2 bit enables triggers to toggle the state of the pin.
1 (R/W)	PX1	PX1 Data Toggle on Trigger. The PORT_TRIG_TGL.PX1 bit enables triggers to toggle the state of the pin.
0 (R/W)	PX0	PX0 Data Toggle on Trigger. The PORT_TRIG_TGL.PX0 bit enables triggers to toggle the state of the pin.

ADSP-2156x PINT Register Descriptions

The Pin Interrupt module (PINT) contains the following registers.

Table 12-29: ADSP-2156x PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register

Table 12-29: ADSP-2156x PINT Register List (Continued)

Name	Description
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

PINT Assign Register

The `PINT_ASSIGN` register controls the pin-to-interrupt request assignment in a byte-wide manner. This register consists of four control bytes that each function as a multiplexer control.

The PINT ports are subdivided into 8-bit half ports, resulting in lower and upper half 8-bit units. Using the multiplexers controlled by the `PINT_ASSIGN` register, the lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINT block. The upper half units can be forwarded to either byte 1 or byte 3 of the PINT block, without further restrictions.

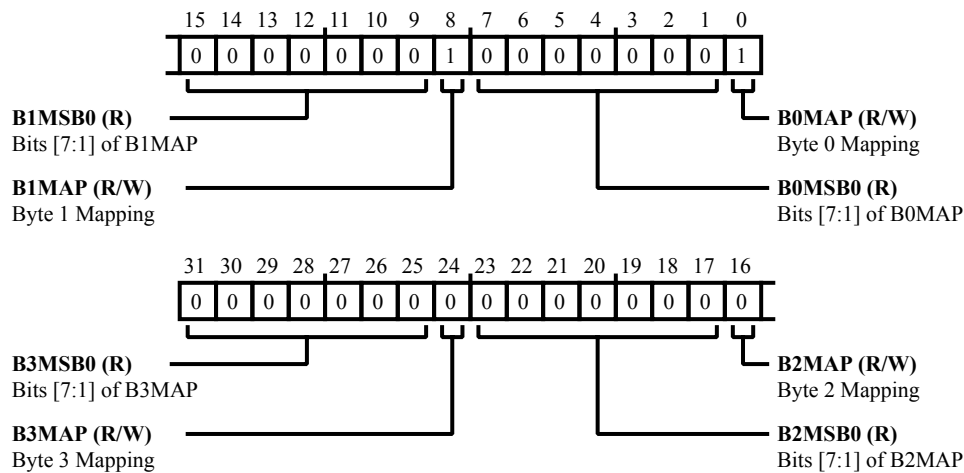


Figure 12-26: PINT_ASSIGN Register Diagram

Table 12-30: PINT_ASSIGN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:25 (R/NW)	B3MSB0	Bits [7:1] of B3MAP.
24 (R/W)	B3MAP	Byte 3 Mapping.
		0 B3MAP_PAH. Byte 3 = PA.H
		1 B3MAP_PBH. Byte 3 = PB.H
23:17 (R/NW)	B2MSB0	Bits [7:1] of B2MAP.
16 (R/W)	B2MAP	Byte 2 Mapping.
		0 B2MAP_PAL. Byte 2 = PA.L
		1 B2MAP_PBL. Byte 2 = PB.L
15:9 (R/NW)	B1MSB0	Bits [7:1] of B1MAP.

Table 12-30: PINT_ASSIGN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	B1MAP	Byte 1 Mapping.	
		0	B1MAP_PAH. Byte 1 = PA.H
		1	B1MAP_PBH. Byte 1 = PB.H
7:1 (R/NW)	B0MSB0	Bits [7:1] of B0MAP.	
0 (R/W)	B0MAP	Byte 0 Mapping.	
		0	B0MAP_PAL. Byte 0 = PA.L
		1	B0MAP_PBL. Byte 0 = PB.L

PINT Edge Clear Register

The `PINT_EDGE_CLR` register permits selecting level-sensitive interrupts. Writing 1 to a bit in `PINT_EDGE_CLR` enables level sensitivity for the corresponding pin interrupt.

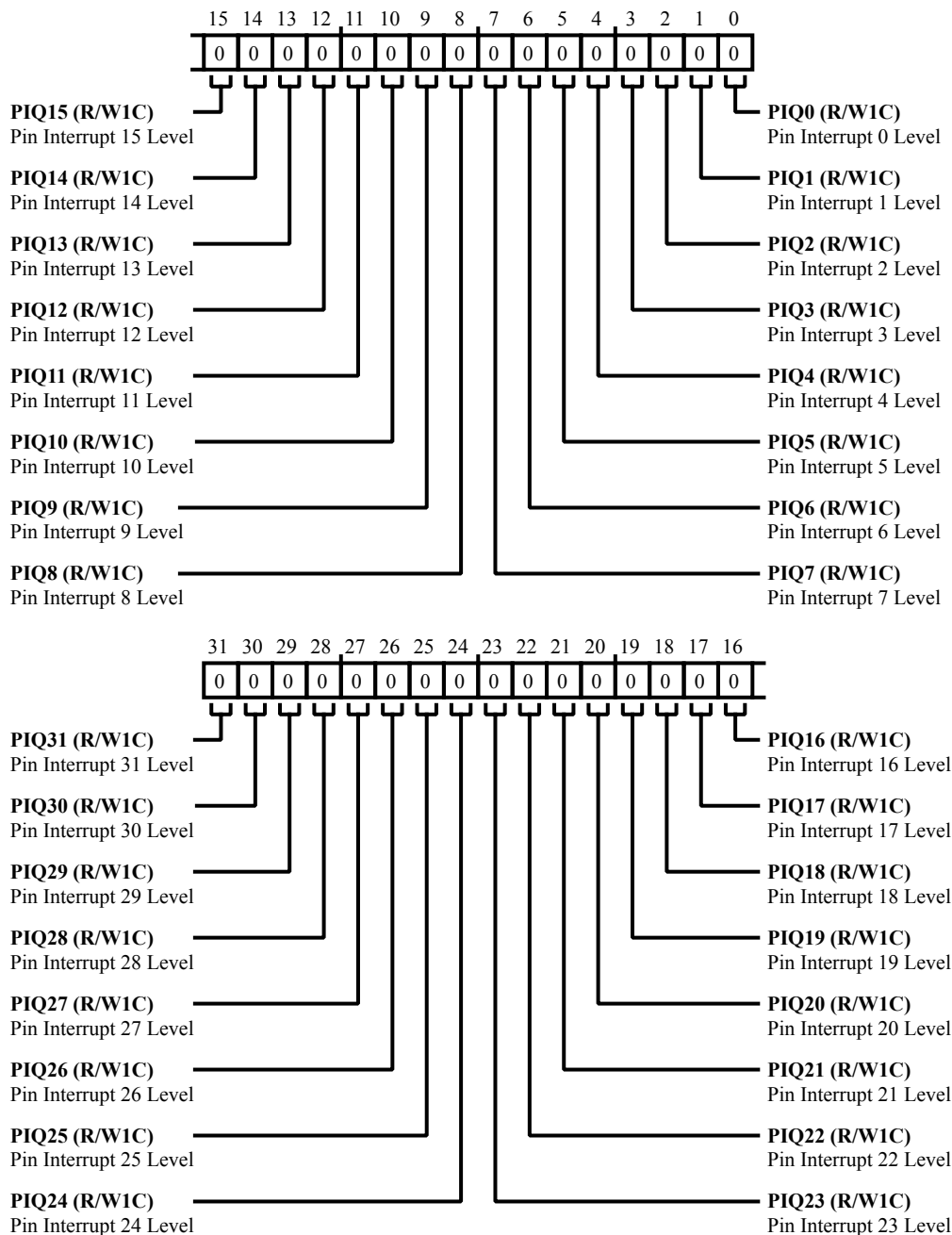


Figure 12-27: `PINT_EDGE_CLR` Register Diagram

Table 12-31: PINT_EDGE_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Level. Set the PINT_EDGE_CLR.PIQ31 bit to enable level sensitivity.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Level. Set the PINT_EDGE_CLR.PIQ30 bit to enable level sensitivity.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Level. Set the PINT_EDGE_CLR.PIQ29 bit to enable level sensitivity.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Level. Set the PINT_EDGE_CLR.PIQ28 bit to enable level sensitivity.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Level. Set the PINT_EDGE_CLR.PIQ27 bit to enable level sensitivity.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Level. Set the PINT_EDGE_CLR.PIQ26 bit to enable level sensitivity.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Level. Set the PINT_EDGE_CLR.PIQ25 bit to enable level sensitivity.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Level. Set the PINT_EDGE_CLR.PIQ24 bit to enable level sensitivity.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Level. Set the PINT_EDGE_CLR.PIQ23 bit to enable level sensitivity.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Level. Set the PINT_EDGE_CLR.PIQ22 bit to enable level sensitivity.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Level. Set the PINT_EDGE_CLR.PIQ21 bit to enable level sensitivity.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Level. Set the PINT_EDGE_CLR.PIQ20 bit to enable level sensitivity.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Level. Set the PINT_EDGE_CLR.PIQ19 bit to enable level sensitivity.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Level. Set the PINT_EDGE_CLR.PIQ18 bit to enable level sensitivity.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Level. Set the PINT_EDGE_CLR.PIQ17 bit to enable level sensitivity.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Level. Set the PINT_EDGE_CLR.PIQ16 bit to enable level sensitivity.

Table 12-31: PINT_EDGE_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Level. Set the PINT_EDGE_CLR.PIQ15 bit to enable level sensitivity.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Level. Set the PINT_EDGE_CLR.PIQ14 bit to enable level sensitivity.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Level. Set the PINT_EDGE_CLR.PIQ13 bit to enable level sensitivity.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Level. Set the PINT_EDGE_CLR.PIQ12 bit to enable level sensitivity.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Level. Set the PINT_EDGE_CLR.PIQ11 bit to enable level sensitivity.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Level. Set the PINT_EDGE_CLR.PIQ10 bit to enable level sensitivity.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Level. Set the PINT_EDGE_CLR.PIQ9 bit to enable level sensitivity.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Level. Set the PINT_EDGE_CLR.PIQ8 bit to enable level sensitivity.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Level. Set the PINT_EDGE_CLR.PIQ7 bit to enable level sensitivity.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Level. Set the PINT_EDGE_CLR.PIQ6 bit to enable level sensitivity.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Level. Set the PINT_EDGE_CLR.PIQ5 bit to enable level sensitivity.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Level. Set the PINT_EDGE_CLR.PIQ4 bit to enable level sensitivity.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Level. Set the PINT_EDGE_CLR.PIQ3 bit to enable level sensitivity.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Level. Set the PINT_EDGE_CLR.PIQ2 bit to enable level sensitivity.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Level. Set the PINT_EDGE_CLR.PIQ1 bit to enable level sensitivity.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Level. Set the PINT_EDGE_CLR.PIQ0 bit to enable level sensitivity.

PINT Edge Set Register

The `PINT_EDGE_SET` register permits selecting edge-sensitive interrupts. Writing 1 to a bit in `PINT_EDGE_SET` enables edge sensitivity for the corresponding pin interrupt.

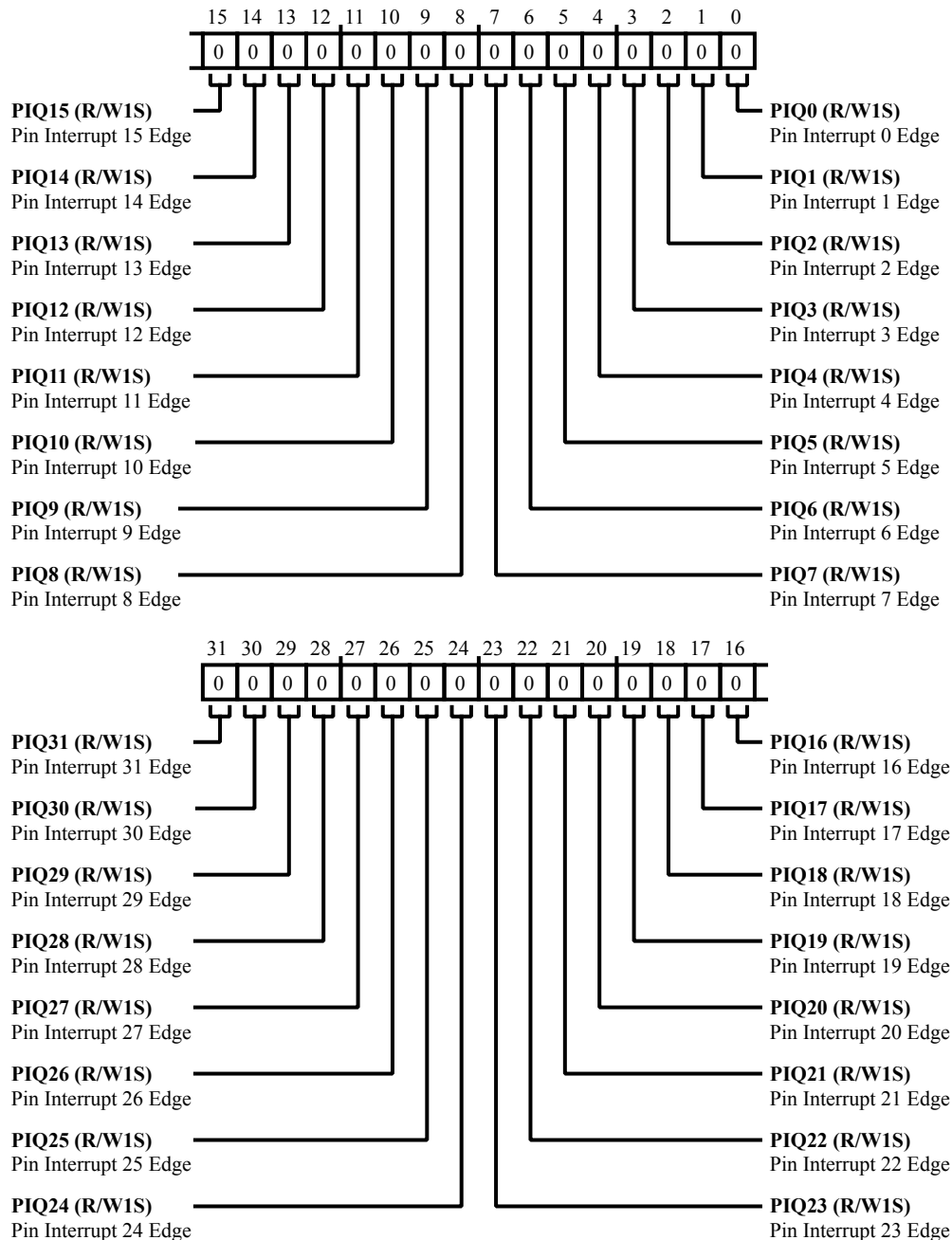


Figure 12-28: `PINT_EDGE_SET` Register Diagram

Table 12-32: PINT_EDGE_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Edge. Set the PINT_EDGE_SET.PIQ31 bit to enable edge sensitivity.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Edge. Set the PINT_EDGE_SET.PIQ30 bit to enable edge sensitivity.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Edge. Set the PINT_EDGE_SET.PIQ29 bit to enable edge sensitivity.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Edge. Set the PINT_EDGE_SET.PIQ28 bit to enable edge sensitivity.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Edge. Set the PINT_EDGE_SET.PIQ27 bit to enable edge sensitivity.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Edge. Set the PINT_EDGE_SET.PIQ26 bit to enable edge sensitivity.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Edge. Set the PINT_EDGE_SET.PIQ25 bit to enable edge sensitivity.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Edge. Set the PINT_EDGE_SET.PIQ24 bit to enable edge sensitivity.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Edge. Set the PINT_EDGE_SET.PIQ23 bit to enable edge sensitivity.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Edge. Set the PINT_EDGE_SET.PIQ22 bit to enable edge sensitivity.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Edge. Set the PINT_EDGE_SET.PIQ21 bit to enable edge sensitivity.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Edge. Set the PINT_EDGE_SET.PIQ20 bit to enable edge sensitivity.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Edge. Set the PINT_EDGE_SET.PIQ19 bit to enable edge sensitivity.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Edge. Set the PINT_EDGE_SET.PIQ18 bit to enable edge sensitivity.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Edge. Set the PINT_EDGE_SET.PIQ17 bit to enable edge sensitivity.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Edge. Set the PINT_EDGE_SET.PIQ16 bit to enable edge sensitivity.

Table 12-32: PINT_EDGE_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Edge. Set the PINT_EDGE_SET.PIQ15 bit to enable edge sensitivity.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Edge. Set the PINT_EDGE_SET.PIQ14 bit to enable edge sensitivity.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Edge. Set the PINT_EDGE_SET.PIQ13 bit to enable edge sensitivity.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Edge. Set the PINT_EDGE_SET.PIQ12 bit to enable edge sensitivity.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Edge. Set the PINT_EDGE_SET.PIQ11 bit to enable edge sensitivity.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Edge. Set the PINT_EDGE_SET.PIQ10 bit to enable edge sensitivity.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Edge. Set the PINT_EDGE_SET.PIQ9 bit to enable edge sensitivity.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Edge. Set the PINT_EDGE_SET.PIQ8 bit to enable edge sensitivity.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Edge. Set the PINT_EDGE_SET.PIQ7 bit to enable edge sensitivity.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Edge. Set the PINT_EDGE_SET.PIQ6 bit to enable edge sensitivity.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Edge. Set the PINT_EDGE_SET.PIQ5 bit to enable edge sensitivity.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Edge. Set the PINT_EDGE_SET.PIQ4 bit to enable edge sensitivity.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Edge. Set the PINT_EDGE_SET.PIQ3 bit to enable edge sensitivity.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Edge. Set the PINT_EDGE_SET.PIQ2 bit to enable edge sensitivity.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Edge. Set the PINT_EDGE_SET.PIQ1 bit to enable edge sensitivity.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Edge. Set the PINT_EDGE_SET.PIQ0 bit to enable edge sensitivity.

PINT Invert Clear Register

The `PINT_INV_CLR` register disables inverting input polarity. Writing 1 to a bit in `PINT_INV_CLR` disables an inverter for input on the corresponding pin.

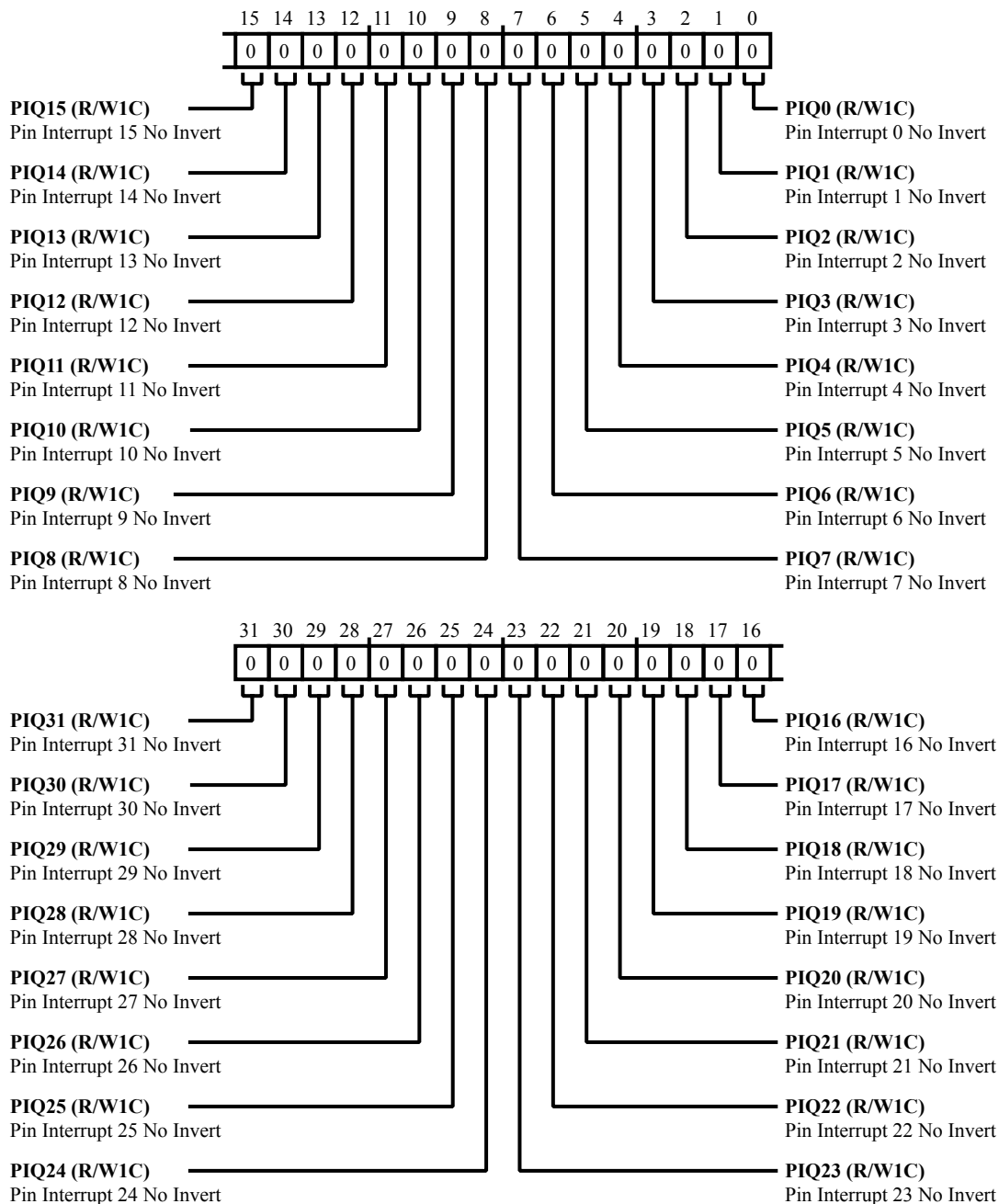


Figure 12-29: PINT_INV_CLR Register Diagram

Table 12-33: PINT_INV_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 No Invert. Set the PINT_INV_CLR.PIQ31 bit to disable inverted input.
30 (R/W1C)	PIQ30	Pin Interrupt 30 No Invert. Set the PINT_INV_CLR.PIQ30 bit to disable inverted input.
29 (R/W1C)	PIQ29	Pin Interrupt 29 No Invert. Set the PINT_INV_CLR.PIQ29 bit to disable inverted input.
28 (R/W1C)	PIQ28	Pin Interrupt 28 No Invert. Set the PINT_INV_CLR.PIQ28 bit to disable inverted input.
27 (R/W1C)	PIQ27	Pin Interrupt 27 No Invert. Set the PINT_INV_CLR.PIQ27 bit to disable inverted input.
26 (R/W1C)	PIQ26	Pin Interrupt 26 No Invert. Set the PINT_INV_CLR.PIQ26 bit to disable inverted input.
25 (R/W1C)	PIQ25	Pin Interrupt 25 No Invert. Set the PINT_INV_CLR.PIQ25 bit to disable inverted input.
24 (R/W1C)	PIQ24	Pin Interrupt 24 No Invert. Set the PINT_INV_CLR.PIQ24 bit to disable inverted input.
23 (R/W1C)	PIQ23	Pin Interrupt 23 No Invert. Set the PINT_INV_CLR.PIQ23 bit to disable inverted input.
22 (R/W1C)	PIQ22	Pin Interrupt 22 No Invert. Set the PINT_INV_CLR.PIQ22 bit to disable inverted input.
21 (R/W1C)	PIQ21	Pin Interrupt 21 No Invert. Set the PINT_INV_CLR.PIQ21 bit to disable inverted input.
20 (R/W1C)	PIQ20	Pin Interrupt 20 No Invert. Set the PINT_INV_CLR.PIQ20 bit to disable inverted input.
19 (R/W1C)	PIQ19	Pin Interrupt 19 No Invert. Set the PINT_INV_CLR.PIQ19 bit to disable inverted input.
18 (R/W1C)	PIQ18	Pin Interrupt 18 No Invert. Set the PINT_INV_CLR.PIQ18 bit to disable inverted input.
17 (R/W1C)	PIQ17	Pin Interrupt 17 No Invert. Set the PINT_INV_CLR.PIQ17 bit to disable inverted input.
16 (R/W1C)	PIQ16	Pin Interrupt 16 No Invert. Set the PINT_INV_CLR.PIQ16 bit to disable inverted input.

Table 12-33: PINT_INV_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 No Invert. Set the PINT_INV_CLR.PIQ15 bit to disable inverted input.
14 (R/W1C)	PIQ14	Pin Interrupt 14 No Invert. Set the PINT_INV_CLR.PIQ14 bit to disable inverted input.
13 (R/W1C)	PIQ13	Pin Interrupt 13 No Invert. Set the PINT_INV_CLR.PIQ13 bit to disable inverted input.
12 (R/W1C)	PIQ12	Pin Interrupt 12 No Invert. Set the PINT_INV_CLR.PIQ12 bit to disable inverted input.
11 (R/W1C)	PIQ11	Pin Interrupt 11 No Invert. Set the PINT_INV_CLR.PIQ11 bit to disable inverted input.
10 (R/W1C)	PIQ10	Pin Interrupt 10 No Invert. Set the PINT_INV_CLR.PIQ10 bit to disable inverted input.
9 (R/W1C)	PIQ9	Pin Interrupt 9 No Invert. Set the PINT_INV_CLR.PIQ9 bit to disable inverted input.
8 (R/W1C)	PIQ8	Pin Interrupt 8 No Invert. Set the PINT_INV_CLR.PIQ8 bit to disable inverted input.
7 (R/W1C)	PIQ7	Pin Interrupt 7 No Invert. Set the PINT_INV_CLR.PIQ7 bit to disable inverted input.
6 (R/W1C)	PIQ6	Pin Interrupt 6 No Invert. Set the PINT_INV_CLR.PIQ6 bit to disable inverted input.
5 (R/W1C)	PIQ5	Pin Interrupt 5 No Invert. Set the PINT_INV_CLR.PIQ5 bit to disable inverted input.
4 (R/W1C)	PIQ4	Pin Interrupt 4 No Invert. Set the PINT_INV_CLR.PIQ4 bit to disable inverted input.
3 (R/W1C)	PIQ3	Pin Interrupt 3 No Invert. Set the PINT_INV_CLR.PIQ3 bit to disable inverted input.
2 (R/W1C)	PIQ2	Pin Interrupt 2 No Invert. Set the PINT_INV_CLR.PIQ2 bit to disable inverted input.
1 (R/W1C)	PIQ1	Pin Interrupt 1 No Invert. Set the PINT_INV_CLR.PIQ1 bit to disable inverted input.
0 (R/W1C)	PIQ0	Pin Interrupt 0 No Invert. Set the PINT_INV_CLR.PIQ0 bit to disable inverted input.

PINT Invert Set Register

The `PINT_INV_SET` register enables inverting input polarity. Writing 1 to a bit in `PINT_INV_SET` enables an inverter for input on the corresponding pin.

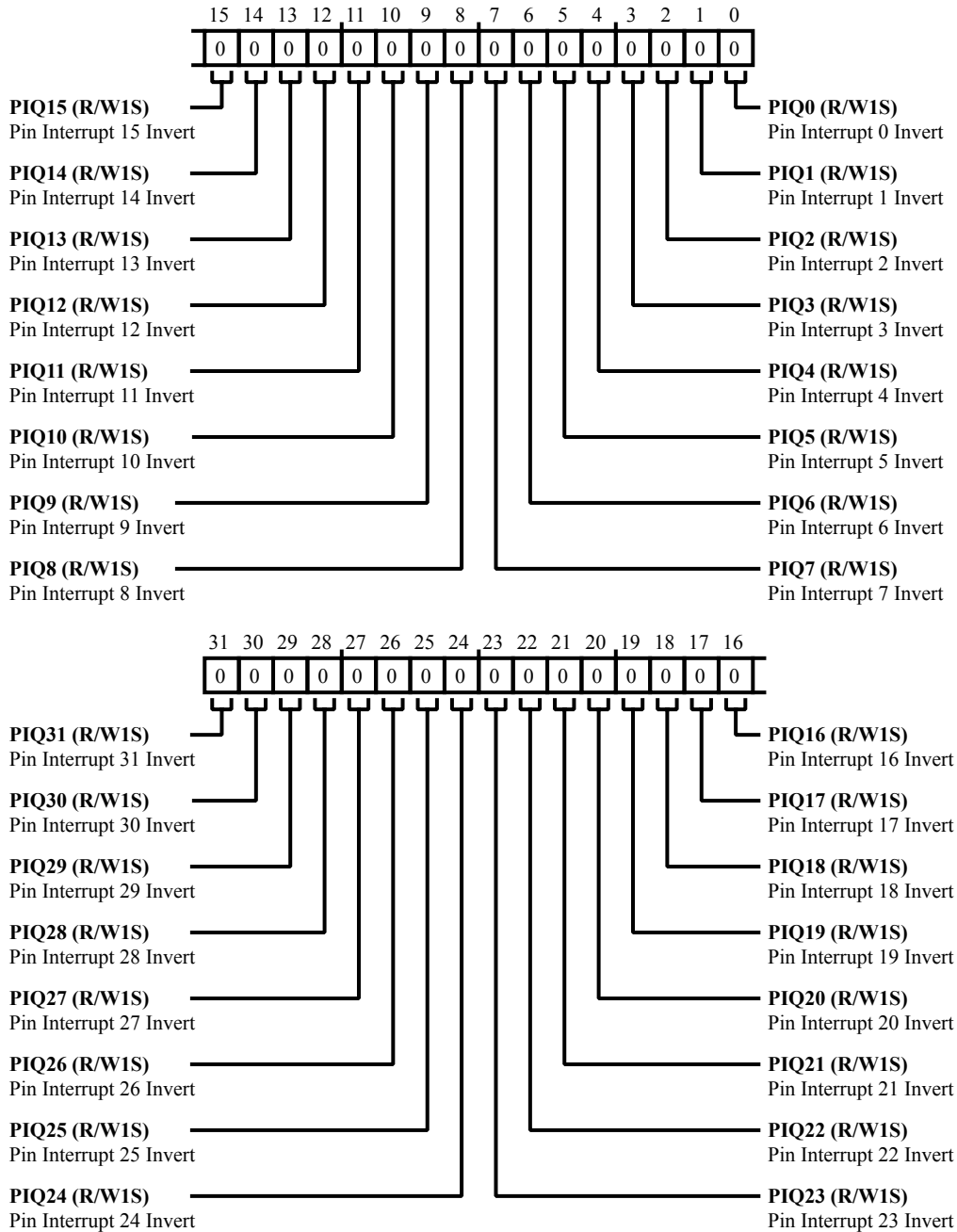


Figure 12-30: PINT_INV_SET Register Diagram

Table 12-34: PINT_INV_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Invert. Set the PINT_INV_SET.PIQ31 bit to enable inverted input.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Invert. Set the PINT_INV_SET.PIQ30 bit to enable inverted input.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Invert. Set the PINT_INV_SET.PIQ29 bit to enable inverted input.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Invert. Set the PINT_INV_SET.PIQ28 bit to enable inverted input.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Invert. Set the PINT_INV_SET.PIQ27 bit to enable inverted input.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Invert. Set the PINT_INV_SET.PIQ26 bit to enable inverted input.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Invert. Set the PINT_INV_SET.PIQ25 bit to enable inverted input.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Invert. Set the PINT_INV_SET.PIQ24 bit to enable inverted input.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Invert. Set the PINT_INV_SET.PIQ23 bit to enable inverted input.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Invert. Set the PINT_INV_SET.PIQ22 bit to enable inverted input.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Invert. Set the PINT_INV_SET.PIQ21 bit to enable inverted input.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Invert. Set the PINT_INV_SET.PIQ20 bit to enable inverted input.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Invert. Set the PINT_INV_SET.PIQ19 bit to enable inverted input.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Invert. Set the PINT_INV_SET.PIQ18 bit to enable inverted input.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Invert. Set the PINT_INV_SET.PIQ17 bit to enable inverted input.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Invert. Set the PINT_INV_SET.PIQ16 bit to enable inverted input.

Table 12-34: PINT_INV_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Invert. Set the PINT_INV_SET.PIQ15 bit to enable inverted input.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Invert. Set the PINT_INV_SET.PIQ14 bit to enable inverted input.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Invert. Set the PINT_INV_SET.PIQ13 bit to enable inverted input.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Invert. Set the PINT_INV_SET.PIQ12 bit to enable inverted input.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Invert. Set the PINT_INV_SET.PIQ11 bit to enable inverted input.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Invert. Set the PINT_INV_SET.PIQ10 bit to enable inverted input.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Invert. Set the PINT_INV_SET.PIQ9 bit to enable inverted input.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Invert. Set the PINT_INV_SET.PIQ8 bit to enable inverted input.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Invert. Set the PINT_INV_SET.PIQ7 bit to enable inverted input.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Invert. Set the PINT_INV_SET.PIQ6 bit to enable inverted input.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Invert. Set the PINT_INV_SET.PIQ5 bit to enable inverted input.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Invert. Set the PINT_INV_SET.PIQ4 bit to enable inverted input.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Invert. Set the PINT_INV_SET.PIQ3 bit to enable inverted input.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Invert. Set the PINT_INV_SET.PIQ2 bit to enable inverted input.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Invert. Set the PINT_INV_SET.PIQ1 bit to enable inverted input.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Invert. Set the PINT_INV_SET.PIQ0 bit to enable inverted input.

PINT Latch Register

The `PINT_LATCH` register indicates the interrupt latch status for pin interrupts. When set, an interrupt request is latched. When cleared, there is no interrupt request latched.

Both the `PINT_REQ` and `PINT_LATCH` registers indicate whether an interrupt request is latched on the respective pin. The `PINT_LATCH` register is a latch that operates regardless of the interrupt masks. Bits of the `PINT_REQ` register depend on the mask register. The `PINT_REQ` register is a logical AND of the `PINT_LATCH` register and the interrupt mask.

Having two separate registers here enables the user to interrogate certain pins in polling mode while others work in interrupt mode. The `PINT_LATCH` registers can be used for edge detection or pin activity detection.

Both registers have W1C behavior. Writing a 1 to either register clears the respective bits in both registers. For interrupt operation, the user may prefer to W1C the `PINT_REQ` register (address still loaded in Px pointer). In polling mode, it might be cleaner to W1C the `PINT_LATCH` register.

Whether in edge-sensitive mode or level-sensitive mode, `PINT_LATCH` bits are never cleared by hardware except at system reset. Even in level-sensitive mode, the `PINT_LATCH` register functions as latch.

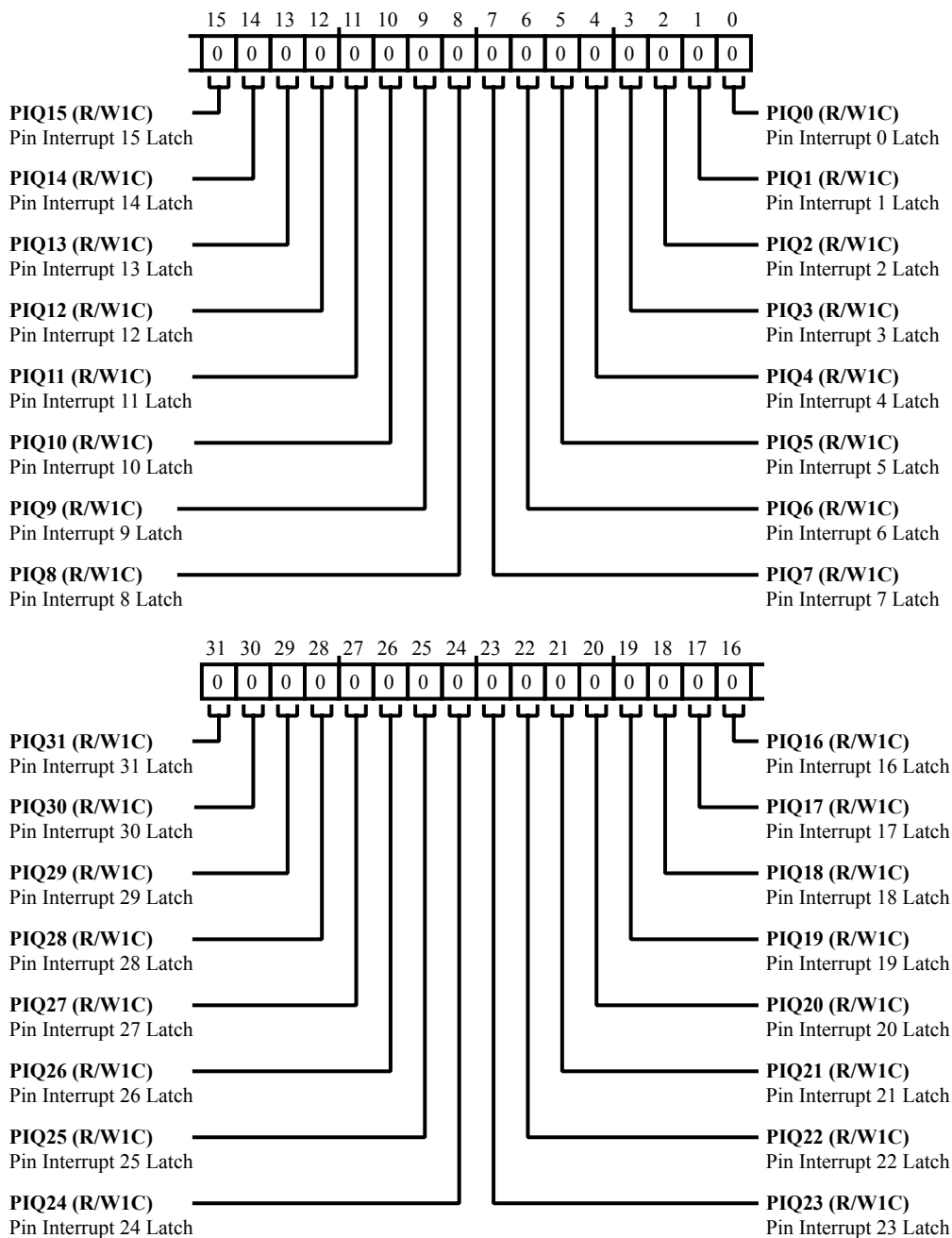


Figure 12-31: PINT_LATCH Register Diagram

Table 12-35: PINT_LATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Latch. If the PINT_LATCH.PIQ31 bit is set, the request is latched.

Table 12-35: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Latch. If the PINT_LATCH.PIQ30 bit is set, the request is latched.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Latch. If the PINT_LATCH.PIQ29 bit is set, the request is latched.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Latch. If the PINT_LATCH.PIQ28 bit is set, the request is latched.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Latch. If the PINT_LATCH.PIQ27 bit is set, the request is latched.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Latch. If the PINT_LATCH.PIQ26 bit is set, the request is latched.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Latch. If the PINT_LATCH.PIQ25 bit is set, the request is latched.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Latch. If the PINT_LATCH.PIQ24 bit is set, the request is latched.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Latch. If the PINT_LATCH.PIQ23 bit is set, the request is latched.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Latch. If the PINT_LATCH.PIQ22 bit is set, the request is latched.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Latch. If the PINT_LATCH.PIQ21 bit is set, the request is latched.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Latch. If the PINT_LATCH.PIQ20 bit is set, the request is latched.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Latch. If the PINT_LATCH.PIQ19 bit is set, the request is latched.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Latch. If the PINT_LATCH.PIQ18 bit is set, the request is latched.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Latch. If the PINT_LATCH.PIQ17 bit is set, the request is latched.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Latch. If the PINT_LATCH.PIQ16 bit is set, the request is latched.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Latch. If the PINT_LATCH.PIQ15 bit is set, the request is latched.

Table 12-35: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Latch. If the PINT_LATCH.PIQ14 bit is set, the request is latched.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Latch. If the PINT_LATCH.PIQ13 bit is set, the request is latched.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Latch. If the PINT_LATCH.PIQ12 bit is set, the request is latched.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Latch. If the PINT_LATCH.PIQ11 bit is set, the request is latched.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Latch. If the PINT_LATCH.PIQ10 bit is set, the request is latched.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Latch. If the PINT_LATCH.PIQ9 bit is set, the request is latched.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Latch. If the PINT_LATCH.PIQ8 bit is set, the request is latched.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Latch. If the PINT_LATCH.PIQ7 bit is set, the request is latched.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Latch. If the PINT_LATCH.PIQ6 bit is set, the request is latched.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Latch. If the PINT_LATCH.PIQ5 bit is set, the request is latched.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Latch. If the PINT_LATCH.PIQ4 bit is set, the request is latched.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Latch. If the PINT_LATCH.PIQ3 bit is set, the request is latched.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Latch. If the PINT_LATCH.PIQ2 bit is set, the request is latched.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Latch. If the PINT_LATCH.PIQ1 bit is set, the request is latched.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Latch. If the PINT_LATCH.PIQ0 bit is set, the request is latched.

PINT Mask Clear Register

The `PINT_MSK_CLR` register permits masking (disabling) of interrupt requests. Writing 1 to a bit in `PINT_MSK_CLR` masks the corresponding pin interrupt.

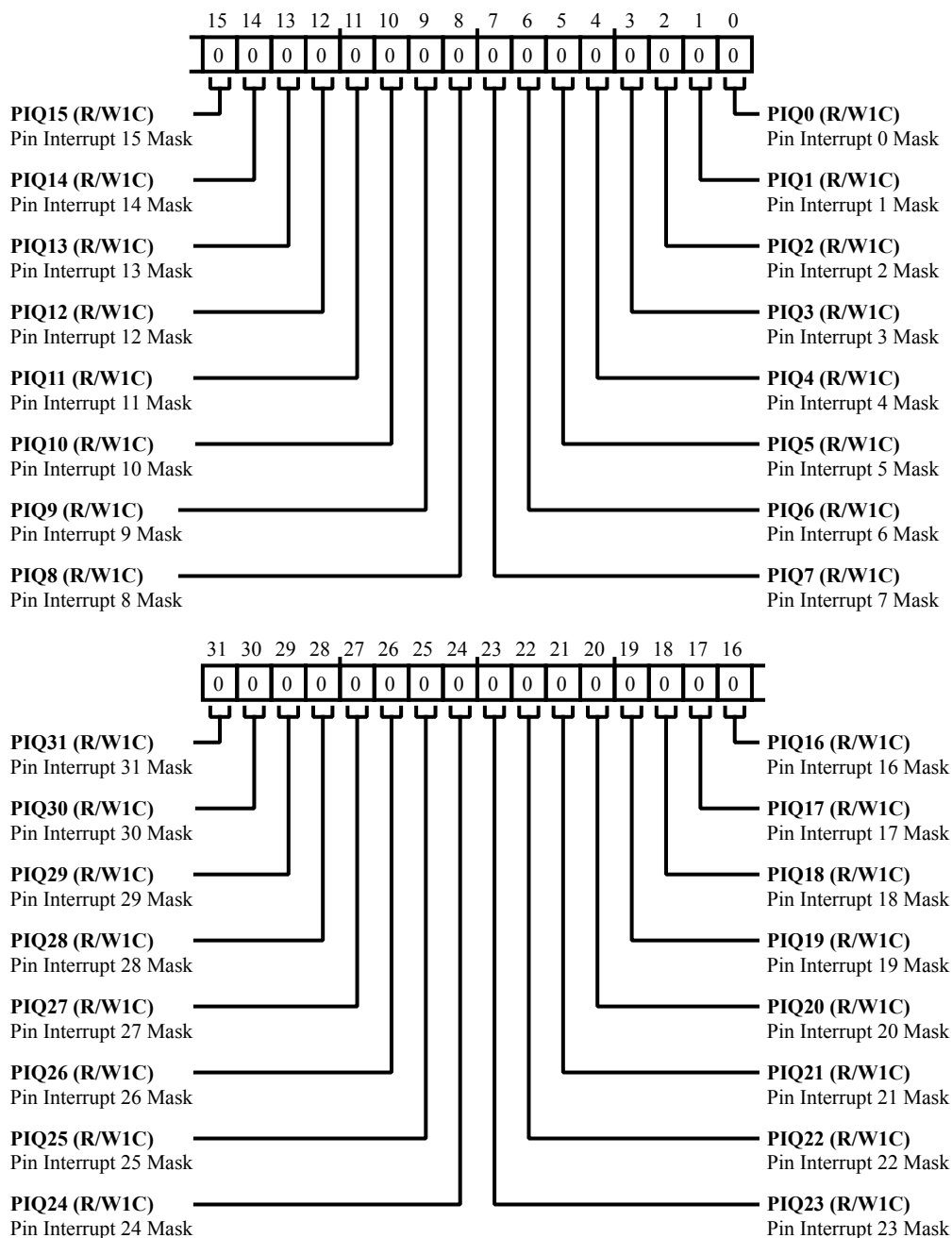


Figure 12-32: PINT_MSK_CLR Register Diagram

Table 12-36: PINT_MSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Mask. Set the PINT_MSK_CLR.PIQ31 bit to disable the interrupt.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Mask. Set the PINT_MSK_CLR.PIQ30 bit to disable the interrupt.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Mask. Set the PINT_MSK_CLR.PIQ29 bit to disable the interrupt.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Mask. Set the PINT_MSK_CLR.PIQ28 bit to disable the interrupt.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Mask. Set the PINT_MSK_CLR.PIQ27 bit to disable the interrupt.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Mask. Set the PINT_MSK_CLR.PIQ26 bit to disable the interrupt.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Mask. Set the PINT_MSK_CLR.PIQ25 bit to disable the interrupt.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Mask. Set the PINT_MSK_CLR.PIQ24 bit to disable the interrupt.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Mask. Set the PINT_MSK_CLR.PIQ23 bit to disable the interrupt.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Mask. Set the PINT_MSK_CLR.PIQ22 bit to disable the interrupt.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Mask. Set the PINT_MSK_CLR.PIQ21 bit to disable the interrupt.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Mask. Set the PINT_MSK_CLR.PIQ20 bit to disable the interrupt.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Mask. Set the PINT_MSK_CLR.PIQ19 bit to disable the interrupt.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Mask. Set the PINT_MSK_CLR.PIQ18 bit to disable the interrupt.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Mask. Set the PINT_MSK_CLR.PIQ17 bit to disable the interrupt.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Mask. Set the PINT_MSK_CLR.PIQ16 bit to disable the interrupt.

Table 12-36: PINT_MSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Mask. Set the PINT_MSK_CLR.PIQ15 bit to disable the interrupt.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Mask. Set the PINT_MSK_CLR.PIQ14 bit to disable the interrupt.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Mask. Set the PINT_MSK_CLR.PIQ13 bit to disable the interrupt.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Mask. Set the PINT_MSK_CLR.PIQ12 bit to disable the interrupt.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Mask. Set the PINT_MSK_CLR.PIQ11 bit to disable the interrupt.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Mask. Set the PINT_MSK_CLR.PIQ10 bit to disable the interrupt.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Mask. Set the PINT_MSK_CLR.PIQ9 bit to disable the interrupt.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Mask. Set the PINT_MSK_CLR.PIQ8 bit to disable the interrupt.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Mask. Set the PINT_MSK_CLR.PIQ7 bit to disable the interrupt.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Mask. Set the PINT_MSK_CLR.PIQ6 bit to disable the interrupt.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Mask. Set the PINT_MSK_CLR.PIQ5 bit to disable the interrupt.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Mask. Set the PINT_MSK_CLR.PIQ4 bit to disable the interrupt.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Mask. Set the PINT_MSK_CLR.PIQ3 bit to disable the interrupt.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Mask. Set the PINT_MSK_CLR.PIQ2 bit to disable the interrupt.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Mask. Set the PINT_MSK_CLR.PIQ1 bit to disable the interrupt.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Mask. Set the PINT_MSK_CLR.PIQ0 bit to disable the interrupt.

PINT Mask Set Register

The `PINT_MSK_SET` register permits unmasking (enabling) of interrupt requests. Writing 1 to a bit in `PINT_MSK_SET` unmask the corresponding pin interrupt.

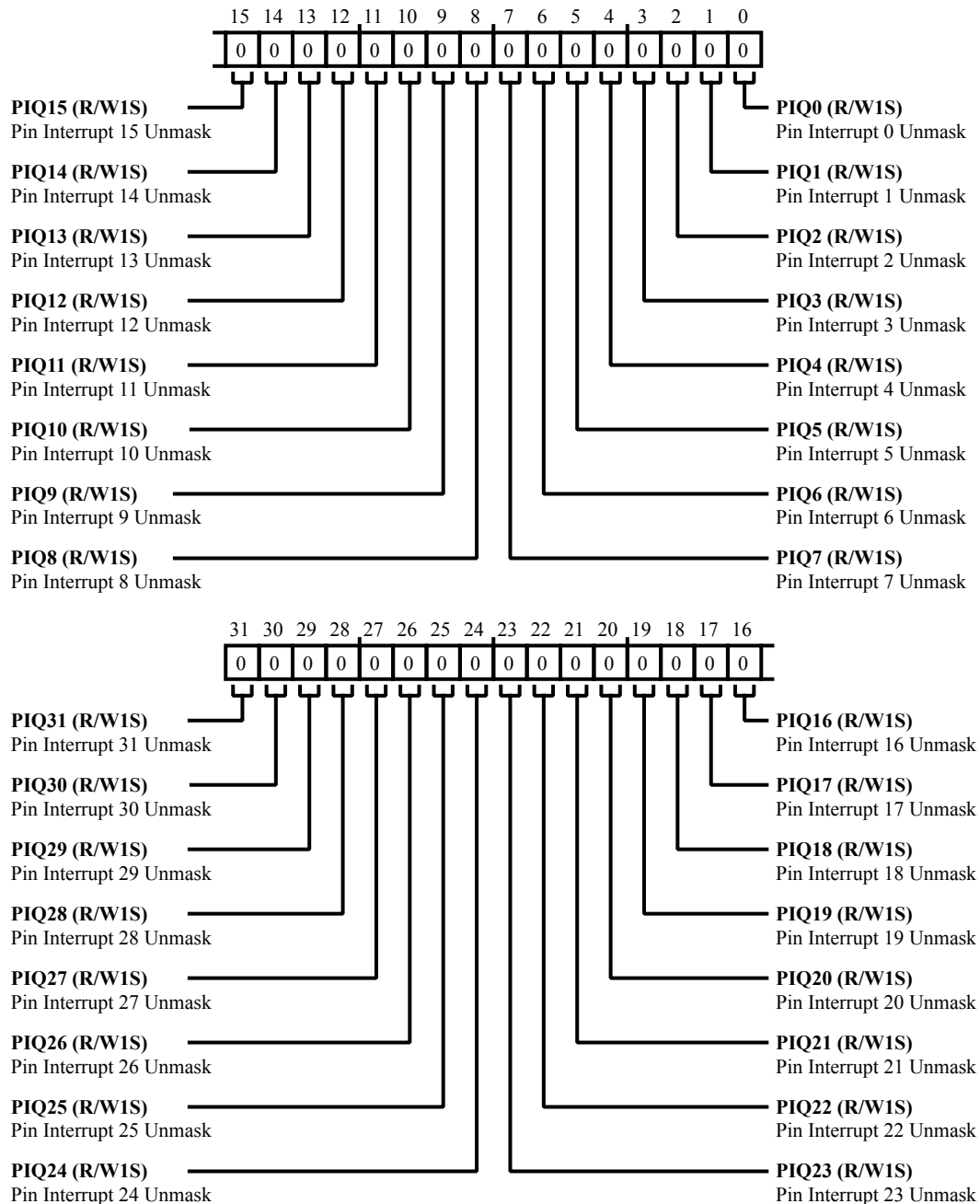


Figure 12-33: PINT_MSK_SET Register Diagram

Table 12-37: PINT_MSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Unmask. Set the PINT_MSK_SET.PIQ31 bit to enable the interrupt.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Unmask. Set the PINT_MSK_SET.PIQ30 bit to enable the interrupt.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Unmask. Set the PINT_MSK_SET.PIQ29 bit to enable the interrupt.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Unmask. Set the PINT_MSK_SET.PIQ28 bit to enable the interrupt.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Unmask. Set the PINT_MSK_SET.PIQ27 bit to enable the interrupt.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Unmask. Set the PINT_MSK_SET.PIQ26 bit to enable the interrupt.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Unmask. Set the PINT_MSK_SET.PIQ25 bit to enable the interrupt.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Unmask. Set the PINT_MSK_SET.PIQ24 bit to enable the interrupt.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Unmask. Set the PINT_MSK_SET.PIQ23 bit to enable the interrupt.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Unmask. Set the PINT_MSK_SET.PIQ22 bit to enable the interrupt.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Unmask. Set the PINT_MSK_SET.PIQ21 bit to enable the interrupt.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Unmask. Set the PINT_MSK_SET.PIQ20 bit to enable the interrupt.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Unmask. Set the PINT_MSK_SET.PIQ19 bit to enable the interrupt.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Unmask. Set the PINT_MSK_SET.PIQ18 bit to enable the interrupt.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Unmask. Set the PINT_MSK_SET.PIQ17 bit to enable the interrupt.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Unmask. Set the PINT_MSK_SET.PIQ16 bit to enable the interrupt.

Table 12-37: PINT_MSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Unmask. Set the PINT_MSK_SET.PIQ15 bit to enable the interrupt.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Unmask. Set the PINT_MSK_SET.PIQ14 bit to enable the interrupt.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Unmask. Set the PINT_MSK_SET.PIQ13 bit to enable the interrupt.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Unmask. Set the PINT_MSK_SET.PIQ12 bit to enable the interrupt.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Unmask. Set the PINT_MSK_SET.PIQ11 bit to enable the interrupt.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Unmask. Set the PINT_MSK_SET.PIQ10 bit to enable the interrupt.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Unmask. Set the PINT_MSK_SET.PIQ9 bit to enable the interrupt.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Unmask. Set the PINT_MSK_SET.PIQ8 bit to enable the interrupt.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Unmask. Set the PINT_MSK_SET.PIQ7 bit to enable the interrupt.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Unmask. Set the PINT_MSK_SET.PIQ6 bit to enable the interrupt.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Unmask. Set the PINT_MSK_SET.PIQ5 bit to enable the interrupt.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Unmask. Set the PINT_MSK_SET.PIQ4 bit to enable the interrupt.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Unmask. Set the PINT_MSK_SET.PIQ3 bit to enable the interrupt.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Unmask. Set the PINT_MSK_SET.PIQ2 bit to enable the interrupt.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Unmask. Set the PINT_MSK_SET.PIQ1 bit to enable the interrupt.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Unmask. Set the PINT_MSK_SET.PIQ0 bit to enable the interrupt.

PINT Pin State Register

When a half port is assigned to a byte in any PINT block, the state of the eight pins (regardless of GPIO or function, input or output) can be seen in the `PINT_PINSTATE` register. While neither input nor output drivers of the pin are enabled, reads of the pin state in `PINT_PINSTATE` return zero. The `PINT_PINSTATE` register reports the inverted state of the pin if the signal inverter is activated by the `PINT_INV_SET` register. The inverter can be enabled on an individual bit-by-bit basis. Every bit in the `PINT_INV_SET` and `PINT_INV_CLR` register pair represents a pin signal.

The pin interrupt pin state registers enable the service routine to read the current state of the pin without reading from GPIO space. If there was an edge-sensitive interrupt, the service routine can check whether the state of the pin is still high or turned low.

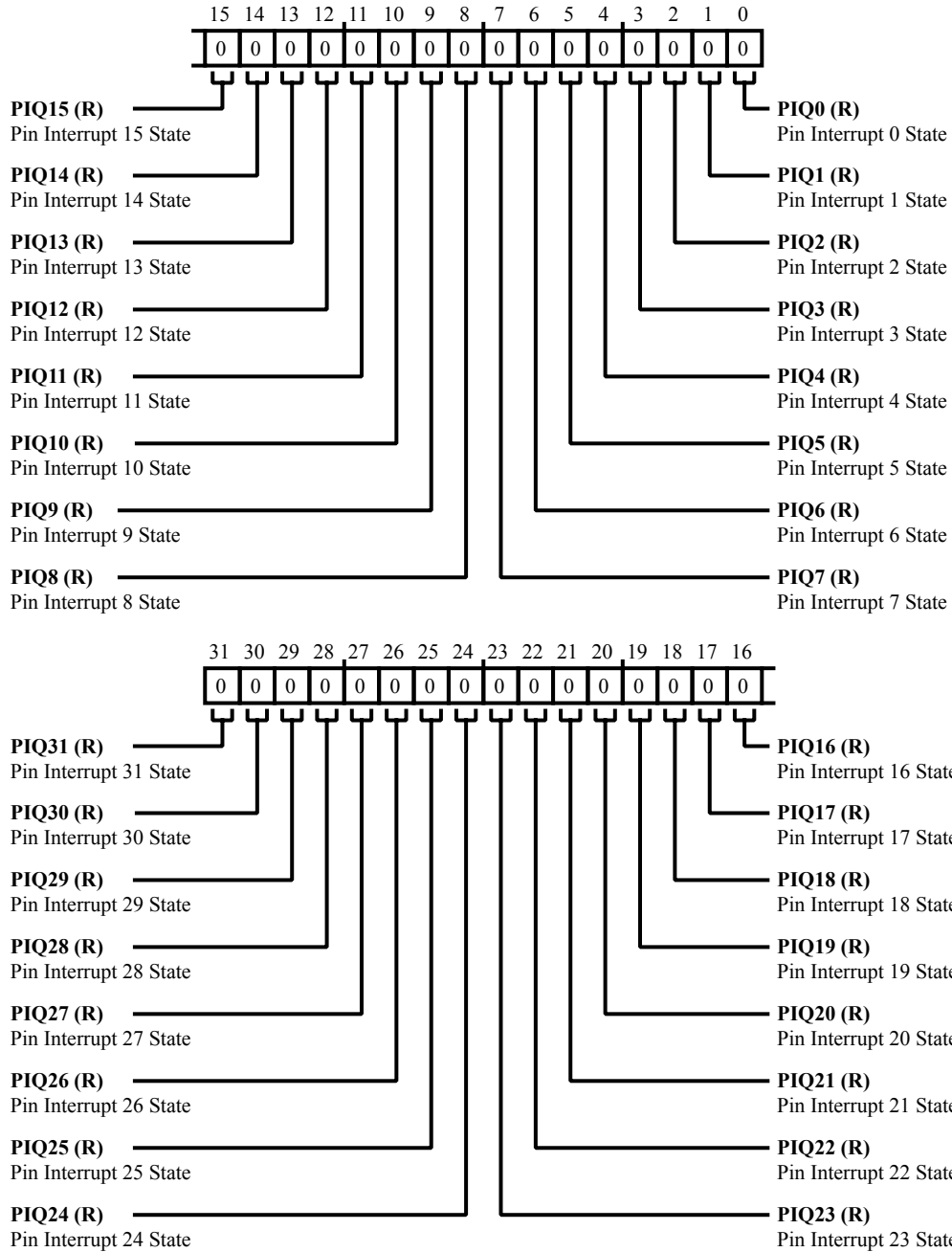


Figure 12-34: PINT_PINSTATE Register Diagram

Table 12-38: PINT_PINSTATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	PIQ31	Pin Interrupt 31 State. A read of the PINT_PINSTATE.PIQ31 bit returns the pin state.

Table 12-38: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	PIQ30	Pin Interrupt 30 State. A read of the PINT_PINSTATE.PIQ30 bit returns the pin state.
29 (R/NW)	PIQ29	Pin Interrupt 29 State. A read of the PINT_PINSTATE.PIQ29 bit returns the pin state.
28 (R/NW)	PIQ28	Pin Interrupt 28 State. A read of the PINT_PINSTATE.PIQ28 bit returns the pin state.
27 (R/NW)	PIQ27	Pin Interrupt 27 State. A read of the PINT_PINSTATE.PIQ27 bit returns the pin state.
26 (R/NW)	PIQ26	Pin Interrupt 26 State. A read of the PINT_PINSTATE.PIQ26 bit returns the pin state.
25 (R/NW)	PIQ25	Pin Interrupt 25 State. A read of the PINT_PINSTATE.PIQ25 bit returns the pin state.
24 (R/NW)	PIQ24	Pin Interrupt 24 State. A read of the PINT_PINSTATE.PIQ24 bit returns the pin state.
23 (R/NW)	PIQ23	Pin Interrupt 23 State. A read of the PINT_PINSTATE.PIQ23 bit returns the pin state.
22 (R/NW)	PIQ22	Pin Interrupt 22 State. A read of the PINT_PINSTATE.PIQ22 bit returns the pin state.
21 (R/NW)	PIQ21	Pin Interrupt 21 State. A read of the PINT_PINSTATE.PIQ21 bit returns the pin state.
20 (R/NW)	PIQ20	Pin Interrupt 20 State. A read of the PINT_PINSTATE.PIQ20 bit returns the pin state.
19 (R/NW)	PIQ19	Pin Interrupt 19 State. A read of the PINT_PINSTATE.PIQ19 bit returns the pin state.
18 (R/NW)	PIQ18	Pin Interrupt 18 State. A read of the PINT_PINSTATE.PIQ18 bit returns the pin state.
17 (R/NW)	PIQ17	Pin Interrupt 17 State. A read of the PINT_PINSTATE.PIQ17 bit returns the pin state.
16 (R/NW)	PIQ16	Pin Interrupt 16 State. A read of the PINT_PINSTATE.PIQ16 bit returns the pin state.
15 (R/NW)	PIQ15	Pin Interrupt 15 State. A read of the PINT_PINSTATE.PIQ15 bit returns the pin state.

Table 12-38: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PIQ14	Pin Interrupt 14 State. A read of the PINT_PINSTATE.PIQ14 bit returns the pin state.
13 (R/NW)	PIQ13	Pin Interrupt 13 State. A read of the PINT_PINSTATE.PIQ13 bit returns the pin state.
12 (R/NW)	PIQ12	Pin Interrupt 12 State. A read of the PINT_PINSTATE.PIQ12 bit returns the pin state.
11 (R/NW)	PIQ11	Pin Interrupt 11 State. A read of the PINT_PINSTATE.PIQ11 bit returns the pin state.
10 (R/NW)	PIQ10	Pin Interrupt 10 State. A read of the PINT_PINSTATE.PIQ10 bit returns the pin state.
9 (R/NW)	PIQ9	Pin Interrupt 9 State. A read of the PINT_PINSTATE.PIQ9 bit returns the pin state.
8 (R/NW)	PIQ8	Pin Interrupt 8 State. A read of the PINT_PINSTATE.PIQ8 bit returns the pin state.
7 (R/NW)	PIQ7	Pin Interrupt 7 State. A read of the PINT_PINSTATE.PIQ7 bit returns the pin state.
6 (R/NW)	PIQ6	Pin Interrupt 6 State. A read of the PINT_PINSTATE.PIQ6 bit returns the pin state.
5 (R/NW)	PIQ5	Pin Interrupt 5 State. A read of the PINT_PINSTATE.PIQ5 bit returns the pin state.
4 (R/NW)	PIQ4	Pin Interrupt 4 State. A read of the PINT_PINSTATE.PIQ4 bit returns the pin state.
3 (R/NW)	PIQ3	Pin Interrupt 3 State. A read of the PINT_PINSTATE.PIQ3 bit returns the pin state.
2 (R/NW)	PIQ2	Pin Interrupt 2 State. A read of the PINT_PINSTATE.PIQ2 bit returns the pin state.
1 (R/NW)	PIQ1	Pin Interrupt 1 State. A read of the PINT_PINSTATE.PIQ1 bit returns the pin state.
0 (R/NW)	PIQ0	Pin Interrupt 0 State. A read of the PINT_PINSTATE.PIQ0 bit returns the pin state.

PINT Request Register

The `PINT_REQ` register indicates the interrupt request status for pin interrupts. When set, an interrupt request is pending. When cleared, there is no interrupt request pending.

Both the `PINT_REQ` and `PINT_LATCH` registers indicate whether an interrupt request is latched on the respective pin. The `PINT_LATCH` register is a latch that operates regardless of the interrupt masks. Bits of the `PINT_REQ` register depend on the mask register. The `PINT_REQ` register is a logical AND of the `PINT_LATCH` register and the interrupt mask.

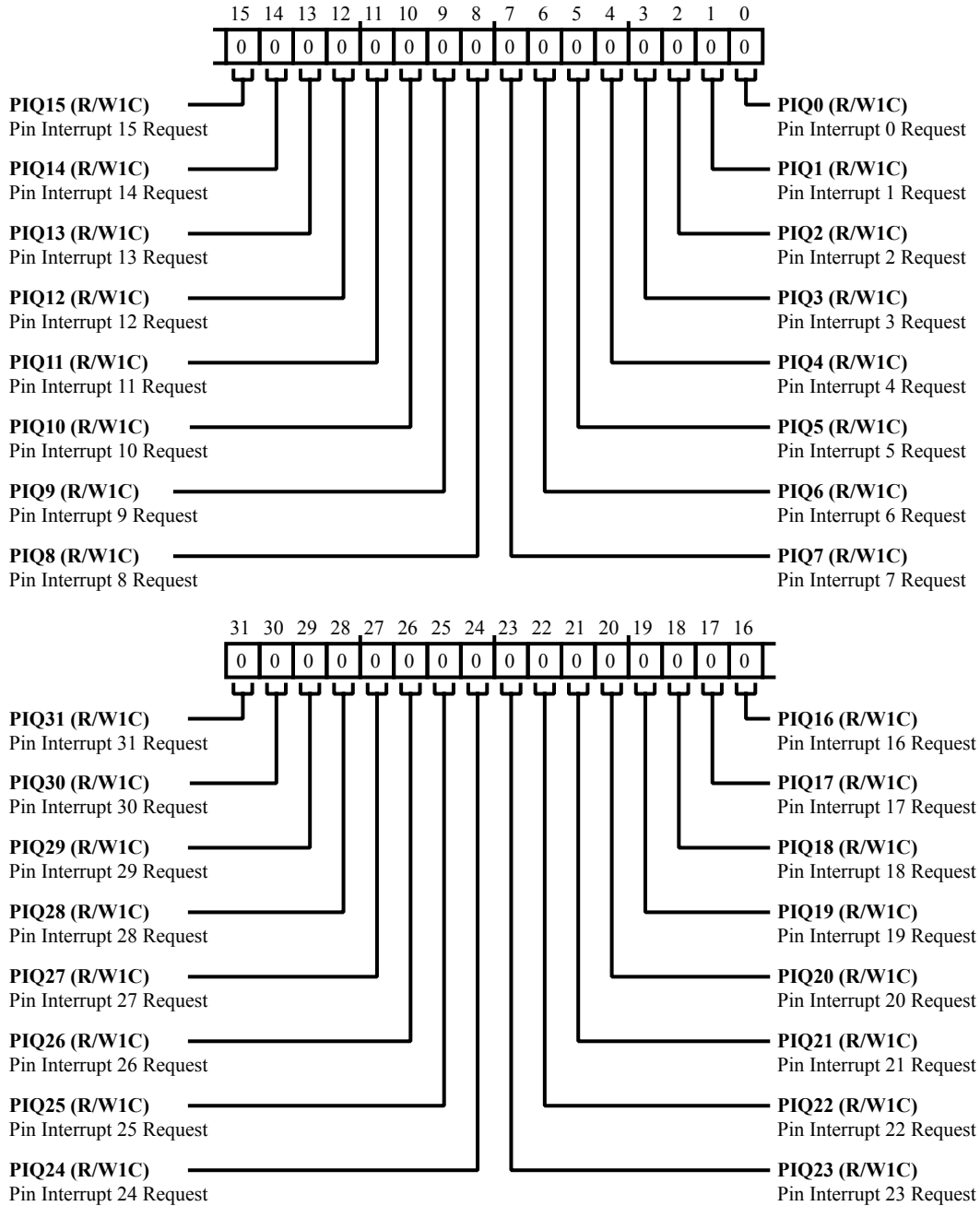


Figure 12-35: PINT_REQ Register Diagram

Table 12-39: PINT_REQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Request. If the PINT_REQ.PIQ31 bit is set, a request is pending.

Table 12-39: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Request. If the PINT_REQ.PIQ30 bit is set, a request is pending.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Request. If the PINT_REQ.PIQ29 bit is set, a request is pending.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Request. If the PINT_REQ.PIQ28 bit is set, a request is pending.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Request. If the PINT_REQ.PIQ27 bit is set, a request is pending.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Request. If the PINT_REQ.PIQ26 bit is set, a request is pending.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Request. If the PINT_REQ.PIQ25 bit is set, a request is pending.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Request. If the PINT_REQ.PIQ24 bit is set, a request is pending.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Request. If the PINT_REQ.PIQ23 bit is set, a request is pending.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Request. If the PINT_REQ.PIQ22 bit is set, a request is pending.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Request. If the PINT_REQ.PIQ21 bit is set, a request is pending.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Request. If the PINT_REQ.PIQ20 bit is set, a request is pending.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Request. If the PINT_REQ.PIQ19 bit is set, a request is pending.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Request. If the PINT_REQ.PIQ18 bit is set, a request is pending.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Request. If the PINT_REQ.PIQ17 bit is set, a request is pending.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Request. If the PINT_REQ.PIQ16 bit is set, a request is pending.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Request. If the PINT_REQ.PIQ15 bit is set, a request is pending.

Table 12-39: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Request. If the PINT_REQ.PIQ14 bit is set, a request is pending.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Request. If the PINT_REQ.PIQ13 bit is set, a request is pending.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Request. If the PINT_REQ.PIQ12 bit is set, a request is pending.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Request. If the PINT_REQ.PIQ11 bit is set, a request is pending.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Request. If the PINT_REQ.PIQ10 bit is set, a request is pending.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Request. If the PINT_REQ.PIQ9 bit is set, a request is pending.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Request. If the PINT_REQ.PIQ8 bit is set, a request is pending.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Request. If the PINT_REQ.PIQ7 bit is set, a request is pending.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Request. If the PINT_REQ.PIQ6 bit is set, a request is pending.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Request. If the PINT_REQ.PIQ5 bit is set, a request is pending.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Request. If the PINT_REQ.PIQ4 bit is set, a request is pending.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Request. If the PINT_REQ.PIQ3 bit is set, a request is pending.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Request. If the PINT_REQ.PIQ2 bit is set, a request is pending.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Request. If the PINT_REQ.PIQ1 bit is set, a request is pending.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Request. If the PINT_REQ.PIQ0 bit is set, a request is pending.

ADSP-2156x PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 12-40: ADSP-2156x PADS Register List

Name	Description
PADS_DAI0_0_DS	DAI0 1 to 10 pins DS control
PADS_DAI0_1_DS	DAI0 11 to 20 pins DS control
PADS_DAI0_IE	DAI0 Port Input Enable Control Register
PADS_DAI1_0_DS	DAI1 1 to 10 pins DS control
PADS_DAI1_1_DS	DAI1 11 to 20 pins DS control
PADS_DAI1_IE	DAI1 Port Input Enable Control Register
PADS_DAI[n]_PDE	DAIx Pull-Down Enable
PADS_DAI[n]_PUE	DAIx Pull-Up Enable
PADS_NONPORTS_DS	Non-GPIO Drive Strength Register
PADS_PCFG0	Peripheral PAD Configuration0 Register
PADS_PORTA0_DS	PORTA 0 to 7 pins DS control
PADS_PORTA1_DS	PORTA 8 - 15 pins DS control
PADS_PORTB0_DS	PORTB 0 to 7 pins DS control
PADS_PORTB1_DS	PORTB 8 - 15 pins DS control
PADS_PORTC0_DS	PORTC 0 to 7 pins DS control
PADS_PORT[n]_PDE	PORTx Pull-Down Enable
PADS_PORT[n]_PUE	PORTx Pull-Up Enable

DAI0 1 to 10 pins DS control

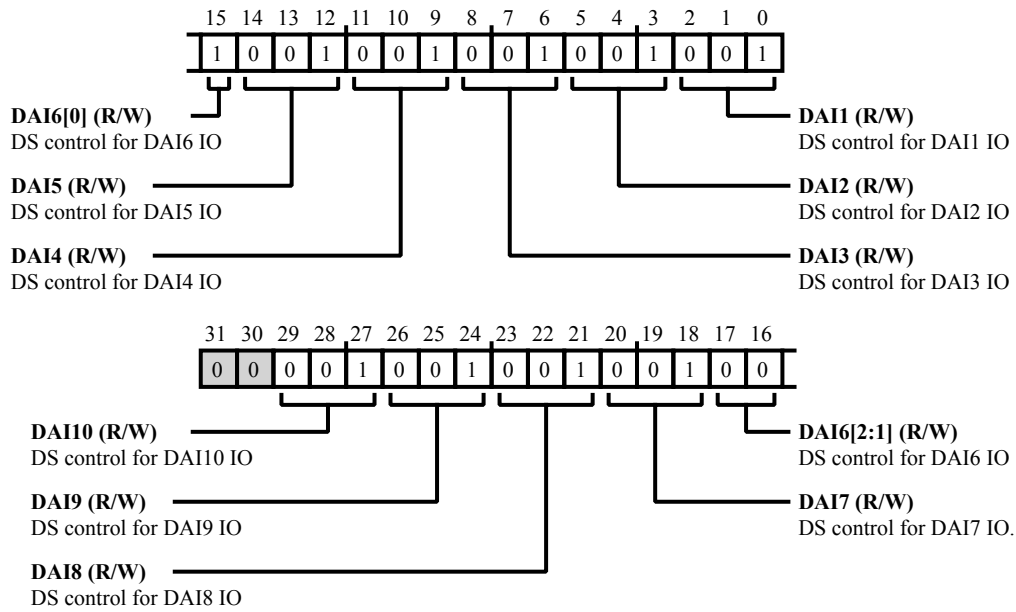


Figure 12-36: PADS_DAI0_0_DS Register Diagram

Table 12-41: PADS_DAI0_0_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:27 (R/W)	DAI10	DS control for DAI10 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
26:24 (R/W)	DAI9	DS control for DAI9 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-41: PADS_DAI0_0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	DAI8	DS control for DAI8 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
20:18 (R/W)	DAI7	DS control for DAI7 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
17:15 (R/W)	DAI6	DS control for DAI6 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	DAI5	DS control for DAI5 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	DAI4	DS control for DAI4 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-41: PADS_DAI0_0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8:6 (R/W)	DAI3	DS control for DAI3 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	DAI2	DS control for DAI2 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
2:0 (R/W)	DAI1	DS control for DAI1 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

DAI0 11 to 20 pins DS control

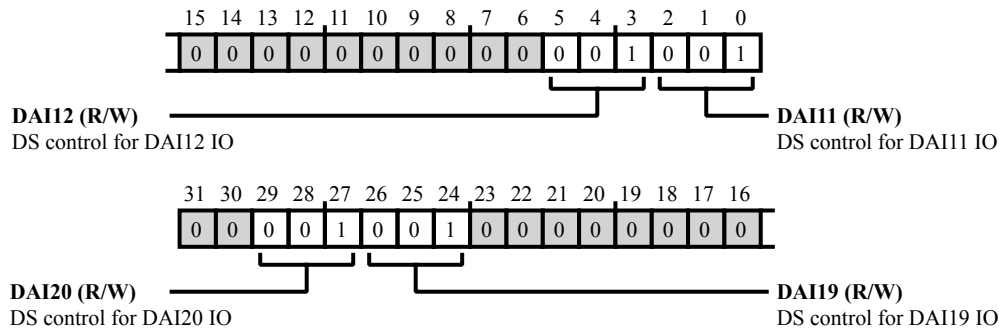


Figure 12-37: PADS_DAI0_1_DS Register Diagram

Table 12-42: PADS_DAI0_1_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:27 (R/W)	DAI20	DS control for DAI20 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
26:24 (R/W)	DAI19	DS control for DAI19 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	DAI12	DS control for DAI12 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-42: PADS_DAI0_1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	DAI11	DS control for DAI11 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

DAI0 Port Input Enable Control Register

The `PADS_DAI0_IE` register configures input enable control of the DAI0 (20 pins) pads. If =0 implies input buffer disable and if =1 implies enable.

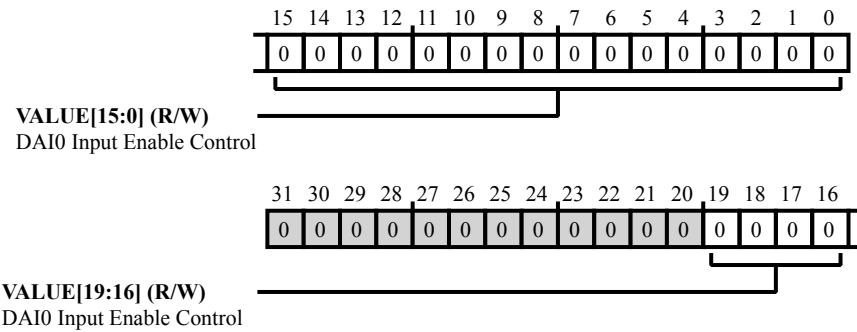


Figure 12-38: PADS_DAI0_IE Register Diagram

Table 12-43: PADS_DAI0_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	VALUE	DAI0 Input Enable Control.

DAI1 1 to 10 pins DS control

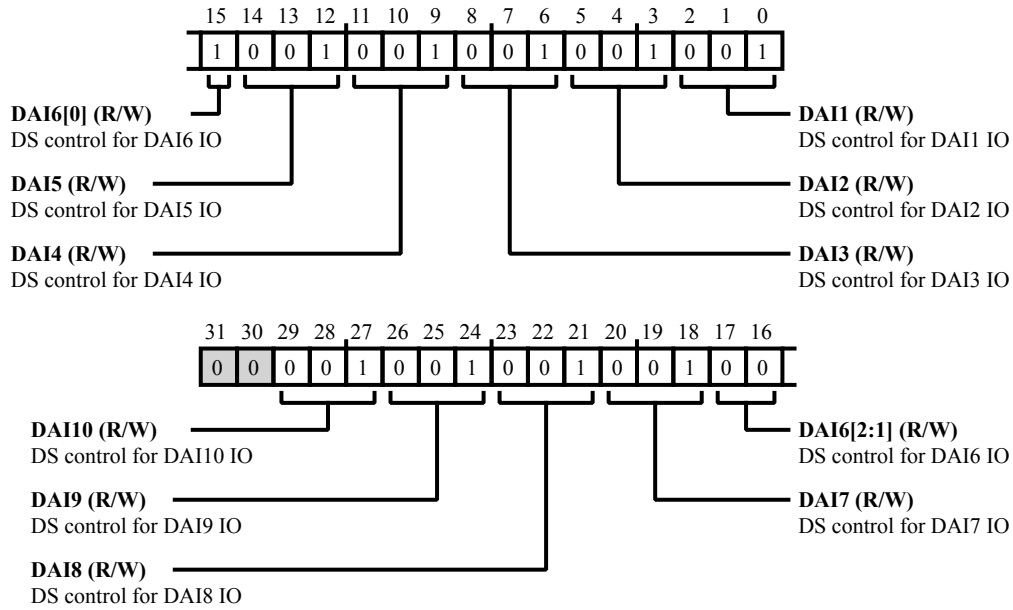


Figure 12-39: PADS_DAI1_0_DS Register Diagram

Table 12-44: PADS_DAI1_0_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:27 (R/W)	DAI10	DS control for DAI10 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
26:24 (R/W)	DAI9	DS control for DAI9 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-44: PADS_DAI1_0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23:21 (R/W)	DAI8	DS control for DAI8 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
20:18 (R/W)	DAI7	DS control for DAI7 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
17:15 (R/W)	DAI6	DS control for DAI6 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	DAI5	DS control for DAI5 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	DAI4	DS control for DAI4 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-44: PADS_DAI1_0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8:6 (R/W)	DAI3	DS control for DAI3 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	DAI2	DS control for DAI2 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
2:0 (R/W)	DAI1	DS control for DAI1 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

DAI1 11 to 20 pins DS control

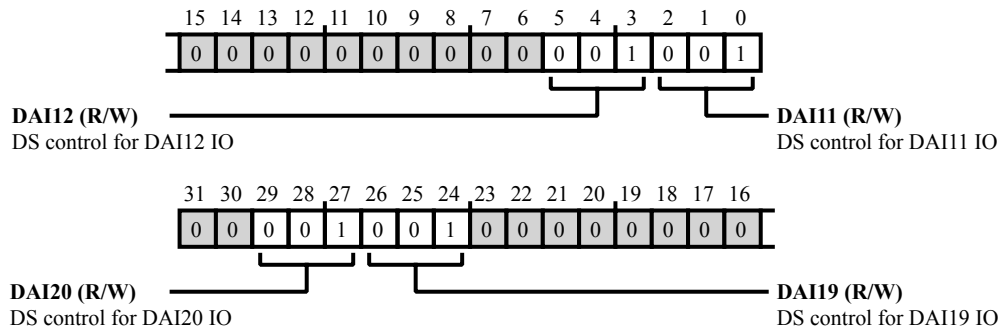


Figure 12-40: PADS_DAI1_1_DS Register Diagram

Table 12-45: PADS_DAI1_1_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:27 (R/W)	DAI20	DS control for DAI20 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
26:24 (R/W)	DAI19	DS control for DAI19 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	DAI12	DS control for DAI12 IO.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-45: PADS_DAI1_1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	DAI11	DS control for DAI11 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

DAI1 Port Input Enable Control Register

The `PADS_DAI1_IE` register configures input enable control of the DAI1 (20 pins) pads. If =0 implies input buffer disable and if =1 implies enable.

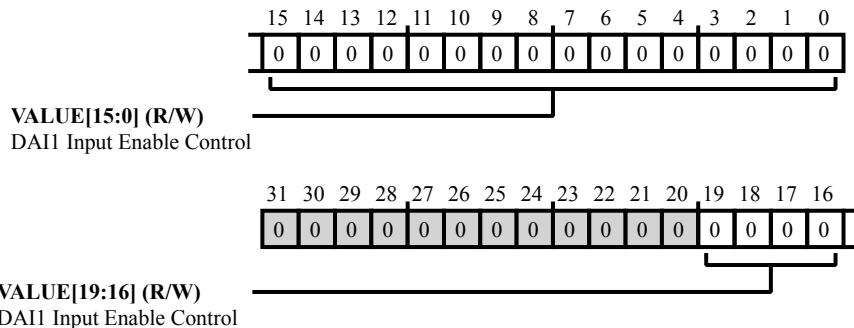


Figure 12-41: PADS_DAI1_IE Register Diagram

Table 12-46: PADS_DAI1_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	VALUE	DAI1 Input Enable Control.

DAIx Pull-Down Enable

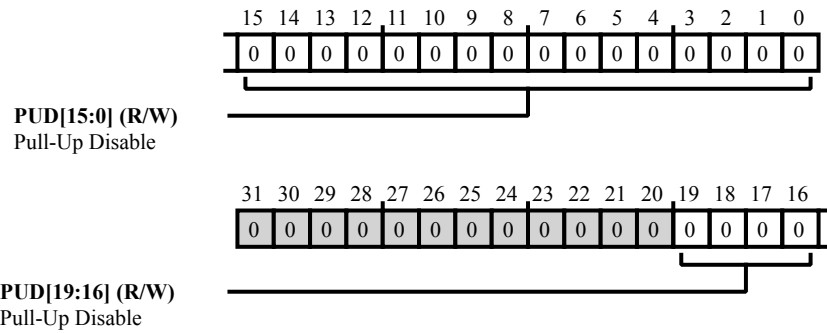


Figure 12-42: PADS_DAI[n]_PDE Register Diagram

Table 12-47: PADS_DAI[n]_PDE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	PUD	Pull-Up Disable.

DAIx Pull-Up Enable

The `PADS_DAI[n]_PUE` register enables the pull-up resistor for DAI pins.

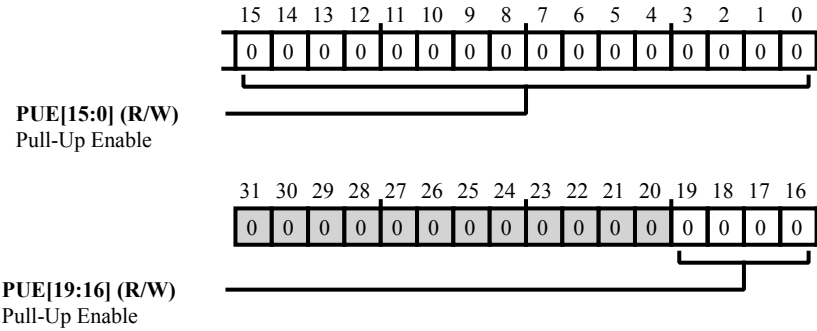


Figure 12-43: PADS_DAI[n]_PUE Register Diagram

Table 12-48: PADS_DAI[n]_PUE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	PUE	Pull-Up Enable.

Non-GPIO Drive Strength Register

The `PADS_NONPORTS_DS` register is for ADI internal usage only. The `PADS_NONPORTS_DS` register sets the drive strength and tolerance for the non-GPIO pins. The drive strength can be controlled only for pin groups, classified by the interface type.

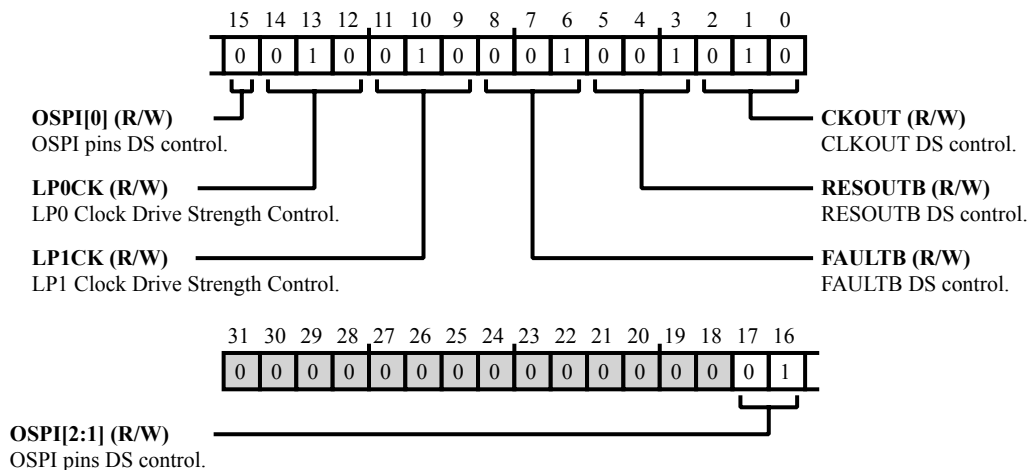


Figure 12-44: `PADS_NONPORTS_DS` Register Diagram

Table 12-49: `PADS_NONPORTS_DS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration		
17:15 (R/W)	OSPI	OSPI pins DS control..		
		<table border="1"> <tr> <td>1</td> <td>Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid</td> </tr> <tr> <td>2</td> <td>Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.</td> </tr> </table>	1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid			
2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.			
14:12 (R/W)	LP0CK	LP0 Clock Drive Strength Control..		
		<table border="1"> <tr> <td>1</td> <td>Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid</td> </tr> <tr> <td>2</td> <td>Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.</td> </tr> </table>	1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid			
2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.			

Table 12-49: PADS_NONPORTS_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11:9 (R/W)	LP1CK	LP1 Clock Drive Strength Control..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	FAULTB	FAULTB DS control..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	RESOUTB	RESOUTB DS control..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
2:0 (R/W)	CKOUT	CLKOUT DS control..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Peripheral PAD Configuration0 Register

The `PADS_PCFG0` register provides several configuration options for the pads and multiplexing for peripherals.

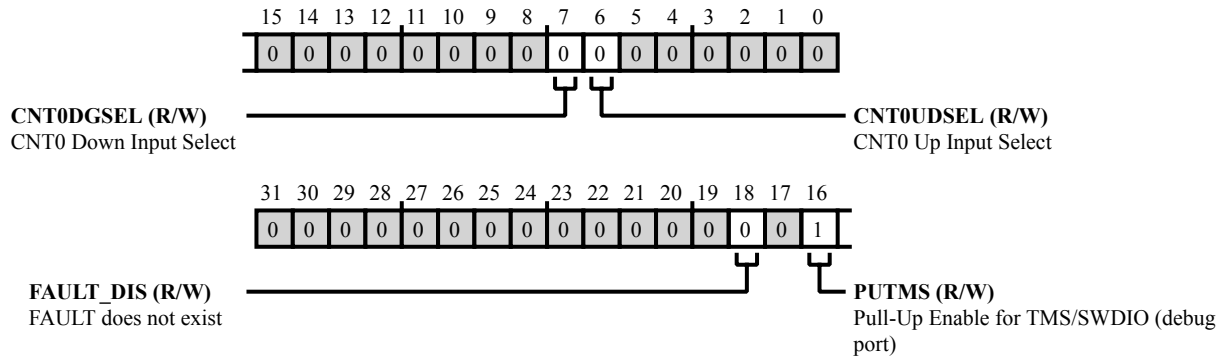


Figure 12-45: PADS_PCFG0 Register Diagram

Table 12-50: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	FAULT_DIS	FAULT does not exist.
		0 Disable input
		1 Enable input
16 (R/W)	PUTMS	Pull-Up Enable for TMS/SWDIO (debug port).
		0 Disable pull-up
		1 Enable pull-up
7 (R/W)	CNT0DGSEL	CNT0 Down Input Select. The <code>PADS_PCFG0</code> bit selects the input for the CNT0 counter down and gate.
		0 GPIO port pins
		1 Trigger connection
6 (R/W)	CNT0UDSEL	CNT0 Up Input Select. The <code>PADS_PCFG0</code> bit selects the input for the CNT0 counter up and direction.
		0 GPIO port pins
		1 Trigger connection

PORTA 0 to 7 pins DS control

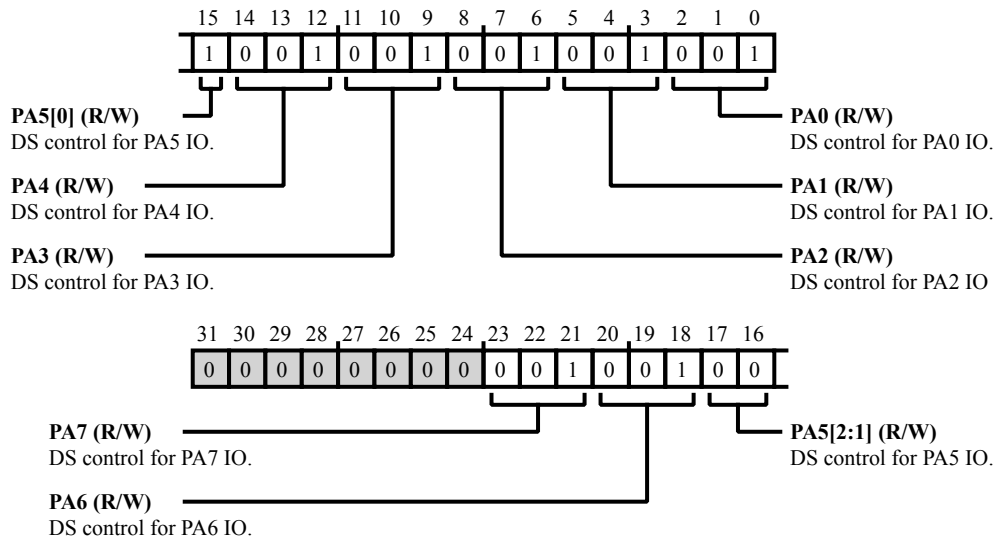


Figure 12-46: PADS_PORTA0_DS Register Diagram

Table 12-51: PADS_PORTA0_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	PA7	DS control for PA7 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
20:18 (R/W)	PA6	DS control for PA6 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-51: PADS_PORTA0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	PA5	DS control for PA5 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	PA4	DS control for PA4 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	PA3	DS control for PA3 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	PA2	DS control for PA2 IO.	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	PA1	DS control for PA1 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-51: PADS_PORTA0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PA0	DS control for PA0 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

PORTA 8 - 15 pins DS control

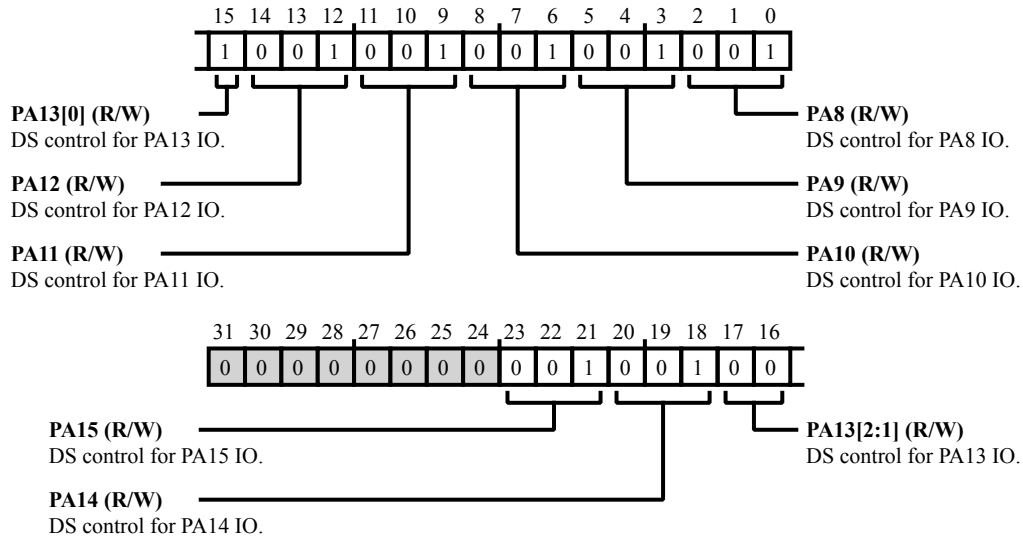


Figure 12-47: PADS_PORTA1_DS Register Diagram

Table 12-52: PADS_PORTA1_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	PA15	DS control for PA15 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
20:18 (R/W)	PA14	2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-52: PADS_PORTA1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	PA13	DS control for PA13 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	PA12	DS control for PA12 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	PA11	DS control for PA11 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	PA10	DS control for PA10 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	PA9	DS control for PA9 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-52: PADS_PORTA1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PA8	DS control for PA8 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

PORTB 0 to 7 pins DS control

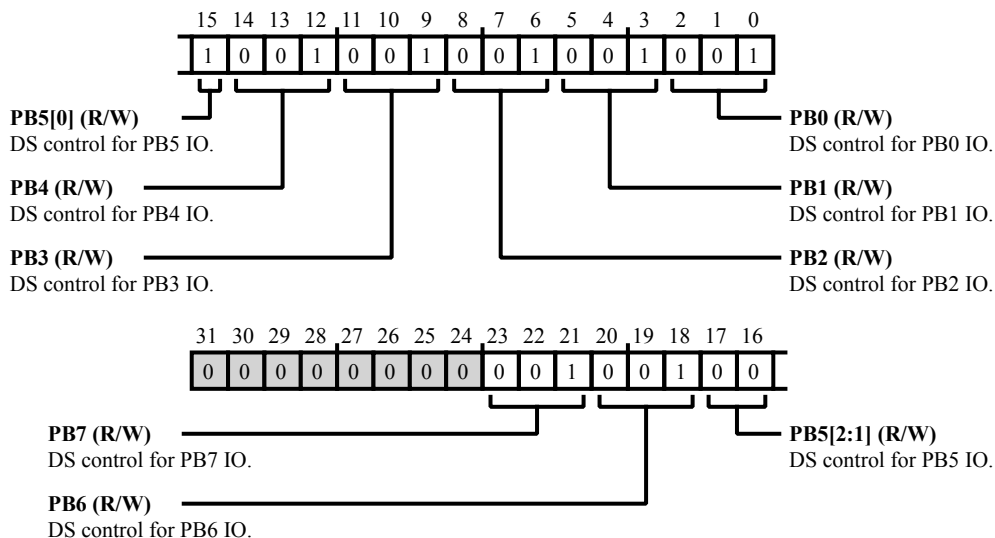


Figure 12-48: PADS_PORTB0_DS Register Diagram

Table 12-53: PADS_PORTB0_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	PB7	DS control for PB7 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
20:18 (R/W)	PB6	DS control for PB6 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-53: PADS_PORTB0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	PB5	DS control for PB5 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	PB4	DS control for PB4 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	PB3	DS control for PB3 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	PB2	DS control for PB2 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	PB1	DS control for PB1 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-53: PADS_PORTB0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PB0	DS control for PB0 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

PORTB 8 - 15 pins DS control

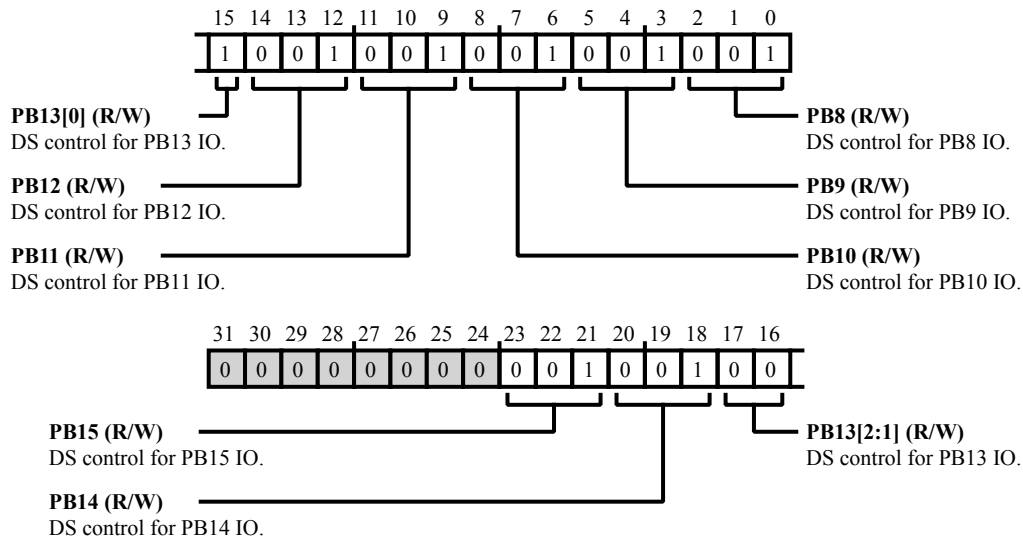


Figure 12-49: PADS_PORTB1_DS Register Diagram

Table 12-54: PADS_PORTB1_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	PB15	DS control for PB15 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
20:18 (R/W)	PB14	DS control for PB14 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-54: PADS_PORTB1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17:15 (R/W)	PB13	DS control for PB13 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	PB12	DS control for PB12 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	PB11	DS control for PB11 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	PB10	DS control for PB10 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	PB9	DS control for PB9 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-54: PADS_PORTB1_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PB8	DS control for PB8 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010. To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

PORTC 0 to 7 pins DS control

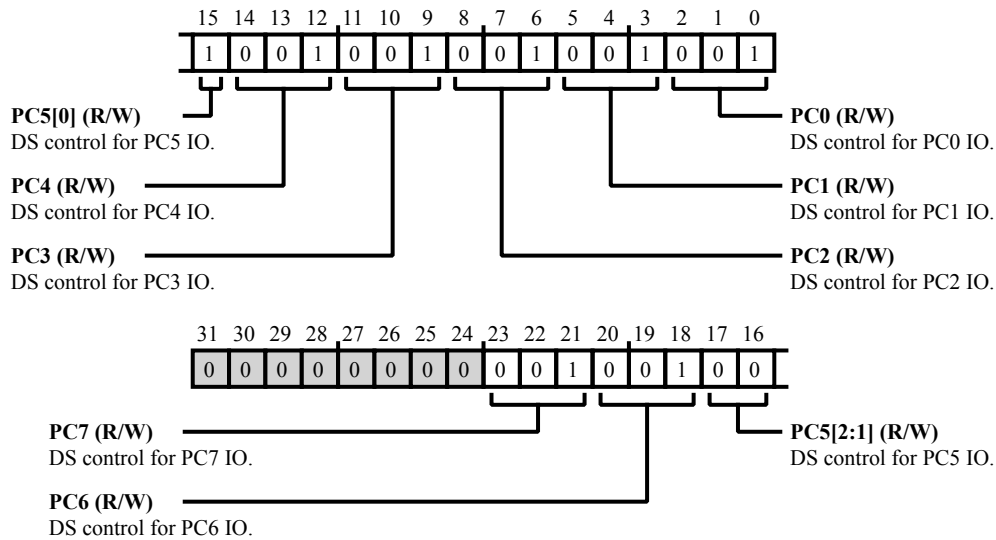


Figure 12-50: PADS_PORTC0_DS Register Diagram

Table 12-55: PADS_PORTC0_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:21 (R/W)	PC7	DS control for PC7 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
20:18 (R/W)	PC6	DS control for PC6 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-55: PADS_PORTC0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17:15 (R/W)	PC5	DS control for PC5 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
14:12 (R/W)	PC4	DS control for PC4 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
11:9 (R/W)	PC3	DS control for PC3 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
8:6 (R/W)	PC2	DS control for PC2 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.
5:3 (R/W)	PC1	DS control for PC1 IO..
		1 Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2 Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

Table 12-55: PADS_PORTC0_DS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PC0	DS control for PC0 IO..	
		1	Drive strength 001. To be used for operating frequency less than or equal to 62.5MHz. Drive Strength 000/1xx are invalid
		2	Drive strength 010.To be used for operating frequency more than 62.5MHz. Drive Strength 000/1xx are invalid.

PORTx Pull-Down Enable

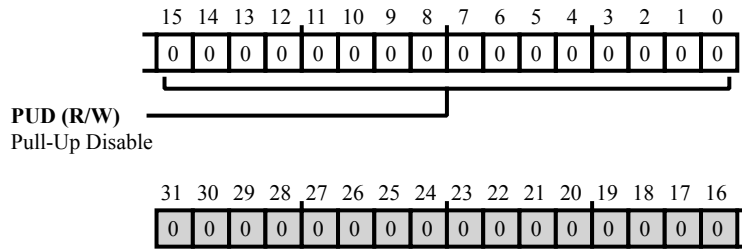


Figure 12-51: PADS_PORT[n]_PDE Register Diagram

Table 12-56: PADS_PORT[n]_PDE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	PUD	Pull-Up Disable.

PORTx Pull-Up Enable

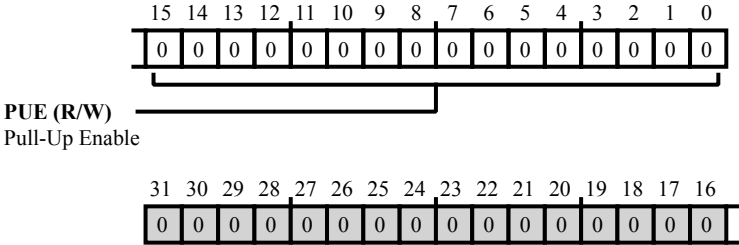


Figure 12-52: PADS_PORT[n]_PUE Register Diagram

Table 12-57: PADS_PORT[n]_PUE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	PUE	Pull-Up Enable.

13 Watchdog Timer (WDOG)

The processor has a 32-bit watchdog timer that can be used to verify system reliability by generating an event to the processor core when the watchdog expires before software updates it.

WDOG Features

The watchdog timer has the following features:

- Programmable 32-bit watchdog count value
- 8-bit disable bit pattern
- General-purpose event generation

The watchdog timer can supervise system software stability by periodically reloading it to prevent expiration of the downward-counting timer (such that the count never becomes 0). When used in this fashion, an expiring timer can indicate the system software is not running normally.

Expiration of the WDOG counter generates a general-purpose interrupt, which can be used in a variety of ways:

- as an interrupt vector sent through the System Event Controller (SEC) to the core to be serviced by a handler function, providing full software control of device resources (for example, GPIO management and reset control).
- as a fault condition through the SEC to provide hardware-automated:
 - signalling of the fault condition on external pins to the system,
 - system reset requests to the Reset Control Unit (RCU), and/or
 - trigger outputs (SEC0_FAULT trigger master) through the Trigger Routing Unit (TRU) to initiate activities in a variety of potential trigger slaves (for example, GPIO control).

WDOG Functional Description

When enabled, the 32-bit watchdog timer counts downward every SCLK0_0 cycle. If it reaches zero, the watchdog expiration event is generated, which can be used in many ways.

To start the watchdog timer:

1. Program the watchdog timeout period (in SCLK0_0 cycles) in the `WDOG_CNT` register. With the watchdog disabled, this write also preloads the `WDOG_STAT` register.
2. Enable the watchdog timer by writing any value other than 0xAD to the `WDOG_CTL.WDEN` field.

Once the watchdog is enabled, writes to the `WDOG_CNT` register are ignored. The counter begins decrementing, and the current counter value can be read from the 32-bit `WDOG_STAT` register at any time.

To prevent the counter from expiring, software must "kick" the watchdog by writing any value to the `WDOG_STAT` register while the current count is non-zero. While the value written is irrelevant and ignored, this action resets the current counter in the `WDOG_STAT` register to the programmed `WDOG_CNT` value, and decrementing continues. The internal counter continues decrementing until it reaches zero, at which point the expiration event is generated, and the `WDOG_CTL.WDRO` rollover bit is set.

Watchdog operation continues in this manner unless disabled by explicitly writing 0xAD to the `WDOG_CTL.WDEN` field.

The watchdog expiration event can be used in a variety of ways, as listed below.

- The watchdog expiration event itself is one of numerous interrupt sources that is managed by the System Event Controller. Like other peripheral sources, this event can be used to cause a vector to a handler function that executes based on interrupt priority.
- The watchdog expiration event can be used to initiate automated hardware response through the SEC Fault Interface (SFI).

For this, the WDOG expiry has to be configured as the fault source in the SEC. Then the response to the WDOG expiry can be set to any of the below:

- Signalling through the external fault pin.
- System reset
- Trigger outputs to numerous potential trigger slaves.

ADSP-2156x WDOG Register List

The Watchdog Timer unit (WDOG) provides a software-based watchdog timer that can improve system reliability by generating an event to the processor core if the watchdog expires before being updated by software. A set of registers governs WDOG operations. For more information on WDOG functionality, see the WDOG register descriptions.

Table 13-1: ADSP-2156x WDOG Register List

Name	Description
<code>WDOG_CNT</code>	Count Register
<code>WDOG_CTL</code>	Control Register
<code>WDOG_STAT</code>	Watchdog Timer Status Register

Table 13-1: ADSP-2156x WDOG Register List (Continued)

Name	Description
WDOG_WIN	Watchdog Timer Window Register

ADSP-2156x WDOG Interrupt List

Table 13-2: ADSP-2156x WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	WDOG0_EXP	WDOG0 Expiration	Level	
4	WDOG1_EXP	WDOG1 Expiration	Level	

WDOG Block Diagram

The *Watchdog Timer Block Diagram* figure shows the detailed block diagram for the watchdog timer.

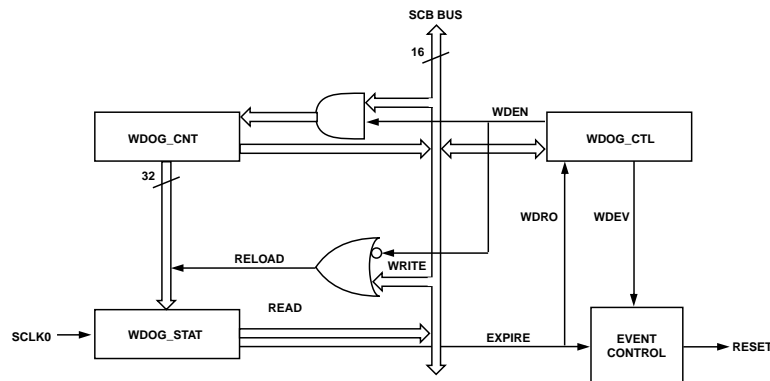


Figure 13-1: Watchdog Timer Block Diagram

Internal Interface

The registers are accessed through the peripheral MMR access bus. 32-bit read/write operations always access the 32-bit `WDOG_CNT` and `WDOG_STAT` registers. Hardware ensures that those accesses are atomic. When the counter expires, the WDOG expiration event is generated.

External Interface

The watchdog timer does not directly interact with any external pins.

ADSP-2156x WDOG Register Descriptions

Watchdog Timer Unit (WDOG) contains the following registers.

Table 13-3: ADSP-2156x WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register
WDOG_WIN	Watchdog Timer Window Register

Count Register

The `WDOG_CNT` register holds the programmable, unsigned count value. A valid write to this register also pre-loads the WDOG counter. For added safety, the `WDOG_CNT` register can be updated only when the WDOG timer is disabled. A write to the `WDOG_CNT` register while the timer is enabled does not modify the contents of this register. This register must be accessed with 32-bit read/writes only.

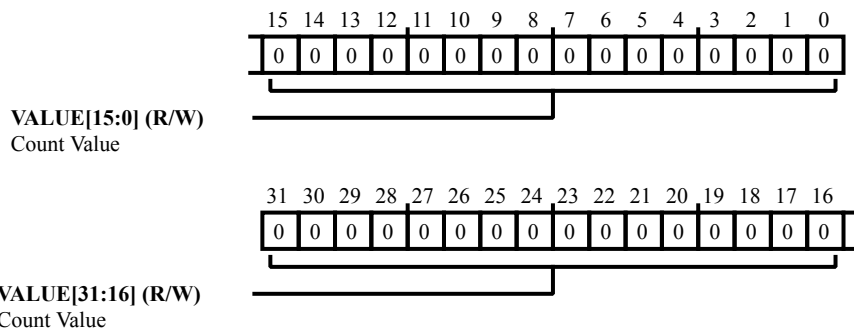


Figure 13-2: WDOG_CNT Register Diagram

Table 13-4: WDOG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count Value. The <code>WDOG_CNT.VALUE</code> bit field holds the programmable, unsigned count value.

Control Register

The `WDOG_CTL` register controls the watchdog timer. This register supports enabling/disabling the watchdog timer and supports checking the timer rollover status. Note that when the processor is in emulation mode, the watchdog timer counter will not decrement even if it is enabled.

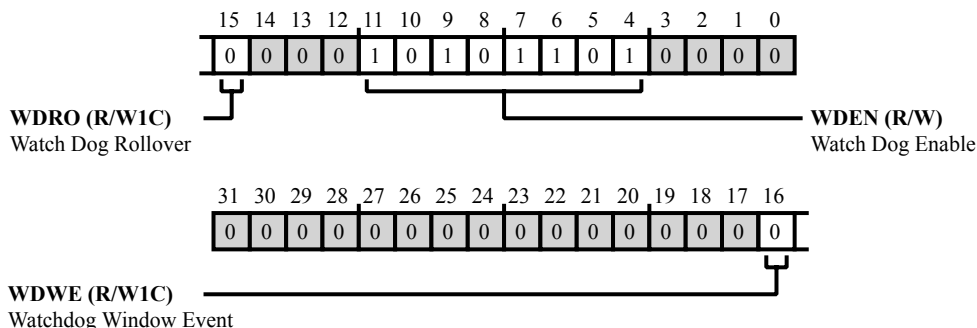


Figure 13-3: WDOG_CTL Register Diagram

Table 13-5: WDOG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W1C)	WDWE	Watchdog Window Event. Software can determine whether the timer has been serviced unexpectedly early by interrogating the <code>WDOG_CTL.WDWE</code> status bit. This is a sticky bit that is set whenever the watchdog is serviced and the <code>WDOG_STAT</code> value is greater than the <code>WDOG_WIN</code> value. It is cleared only by disabling the watchdog timer and by writing a 1 to the bit.
		0 Window Event has not occurred 1 Window Event has occurred
15 (R/W1C)	WDRO	Watch Dog Rollover. Software can determine whether the timer has rolled over by interrogating the <code>WDOG_CTL.WDRO</code> status bit. This is a sticky bit that is set whenever the watch dog timer count reaches 0 and cleared only by disabling the watch dog timer and then writing a 1 to the bit.
		0 WDT Has Not Expired 1 WDT Has Expired
11:4 (R/W)	WDEN	Watch Dog Enable. The <code>WDOG_CTL.WDEN</code> field is used to enable and disable the watchdog timer. Writing any value other than the disable value into this field enables the watchdog timer. This multi-bit disable key minimizes the chance of inadvertently disabling the watchdog timer.
		173 Counter Disabled. All other values mean that the counter is enabled.

Watchdog Timer Status Register

The `WDOG_STAT` register contains the current count value of the watchdog timer. Reads of this register return the current count value. When the watchdog timer is enabled, the `WDOG_STAT` register is decremented by 1 on each `SCLK` cycle. When the count value reaches 0, the watchdog timer stops counting, and the expiry event is generated. The `WDOG_STAT` register is a 32-bit unsigned system MMR that must be accessed with 32-bit reads and writes.

Values cannot be stored directly in this register but are instead copied from the `WDOG_CNT` register. This copy process can happen in two ways:

- While the watchdog timer is disabled, writing the `WDOG_CNT` register pre-loads the `WDOG_STAT` register.
- While the watchdog timer is enabled, writing the `WDOG_STAT` register loads it with the value in the `WDOG_CNT` register.
- While the watchdog timer is disabled, writing to the `WDOG_STAT` register also reloads it with the value in the `WDOG_CNT` register.

When the processor executes a write (of an arbitrary value) to the `WDOG_STAT` register, the value in the `WDOG_CNT` register is copied into the `WDOG_STAT` register. Typically, software sets the value of `WDOG_CNT` at initialization, then periodically writes to the `WDOG_STAT` register before the watchdog timer expires. This reloads the watchdog timer with the value from `WDOG_CNT` and prevents generation of the expiry event.

If the program does not reload the counter before `SCLK` x count register cycles, an expiry event is generated, and the `WDOG_CTL.WDRO` bit is set. When this happens, the counter stops decrementing and remains at zero. If the counter is enabled with a zero loaded to the counter, the `WDOG_CTL.WDRO` bit is set immediately and the counter remains at zero and does not decrement.

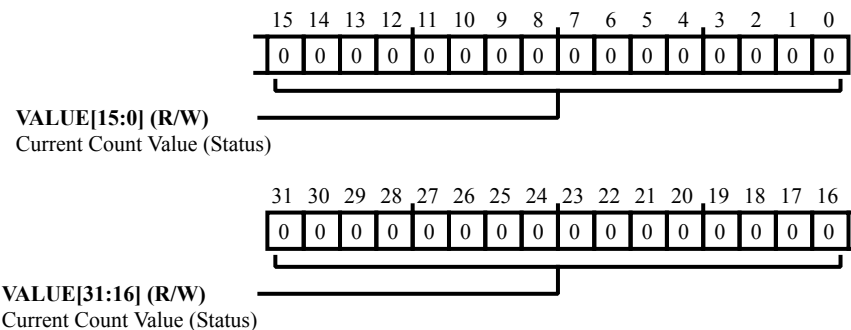


Figure 13-4: `WDOG_STAT` Register Diagram

Table 13-6: WDOG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Current Count Value (Status). The WDOG_STAT.VALUE bit field contains the current count value of the watchdog timer.

Watchdog Timer Window Register

Watchdog window register holds the unsigned window value programmed. The window register can be programmed while the WDOG is disabled; any write to the register when WDOG is enabled doesn't alter the contents of the register. When the WDOG is enabled and the core services the WDOG by doing a write to `WDOG_STAT` register while the `WDOG_STAT` value is greater than `WDOG_WIN`, the WDWE event is generated and also the counter stops decrementing.

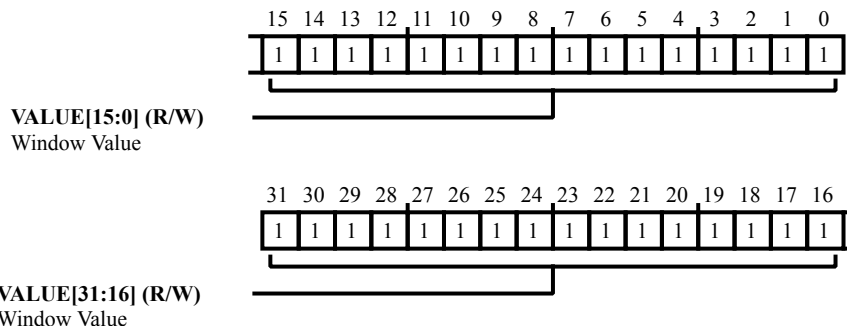


Figure 13-5: WDOG_WIN Register Diagram

Table 13-7: WDOG_WIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Window Value. The <code>WDOG_WIN.VALUE</code> bit field contains the unsigned window value.

14 Link Port (LP)

Link ports allow the processor to connect to other processors or peripheral link ports using a simple communication protocol for high-speed parallel data transfer. This peripheral allows various I/O peripheral interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes.

The link ports of the processor support 8-bit wide data transfers. The link port pins are multiplexed in the GPIO ports. For information on processor multiplexing, see the data sheet for the specific processor.

Link ports can operate independently and simultaneously, allowing glueless high-speed connectivity of up to four external processors.

NOTE: Reference to SCLK in this chapter can be considered as OCLK0_0.

LP Features

All link ports are identical in their design and have the following common features.

- Bidirectional ports with eight data signals (LP_D0 - LP_D7), an acknowledge signal (LP_ACK), and a clock signal (LP_CLK).
- Provide high-speed, point-to-point data transfers to other processors, allowing different types of interconnections between multiple processors.
- Pack data into 32-bit words. The processor can directly read this data or transfer it through DMA to or from on-chip memory.
- Support for data buffering through a 2-deep FIFO for transmit and a 4-deep FIFO for receive.
- Programmable clock and acknowledge based handshake mechanism for efficient communication.
- A dedicated DMA channel.
-

LP Functional Description

This section provides a description of the link port, including a list of its registers and a functional block diagram.

ADSP-2156x LP Register List

The Link Port LP is an 8-bit wide parallel port that can connect to another processor's LP or another LP-compatible device. This port allows a variety of interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes. A set of registers governs LP operations. For more information on LP functionality, see the LP register descriptions.

Table 14-1: ADSP-2156x LP Register List

Name	Description
LP_CTL	Control Register
LP_DIV	Clock Divider Value Register
LP_RX	Receive Buffer Register
LP_STAT	Status Register
LP_TX	Transmit Buffer Register
LP_TXIN_SHDW	Shadow Input Transmit Buffer Register
LP_TXOUT_SHDW	Shadow Output Transmit Buffer Register

ADSP-2156x LP Interrupt List

Table 14-2: ADSP-2156x LP Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
81	LP0_DMA	LP0 DMA Channel		30
82	LP0_STAT	LP0 Status		
83	LP1_DMA	LP1 DMA Channel		36
84	LP1_STAT	LP1 Status		
168	LP0_DMA_ERR	LP0 DMA Data Error		
169	LP1_DMA_ERR	LP1 DMA Data Error		

ADSP-2156x LP Trigger List

Table 14-3: ADSP-2156x LP Trigger List Masters

Trigger ID	Name	Description	Sensitivity
41	LP0_DMA	LP0 DMA Channel	
42	LP1_DMA	LP1 DMA Channel	

Table 14-4: ADSP-2156x LP Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
42	LP0_DMA	LP0 DMA Channel	Pulse
43	LP1_DMA	LP1 DMA Channel	Pulse

ADSP-2156x LP DMA Channel List

Table 14-5: ADSP-2156x LP DMA Channel List

DMA ID	DMA Channel Name	Description
DMA30	LP0_DMA	LP0 DMA Channel
DMA36	LP1_DMA	LP1 DMA Channel

Block Diagram

The *Link Port Block Diagram* shows the block diagram of a link port.

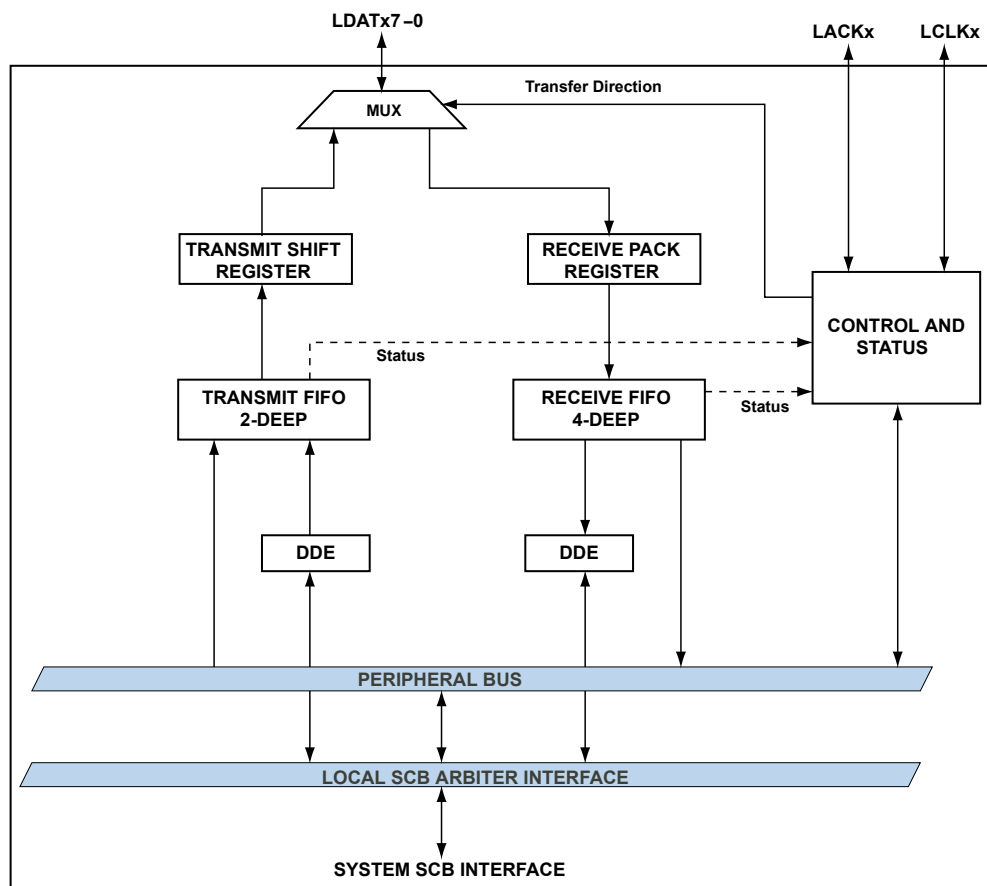
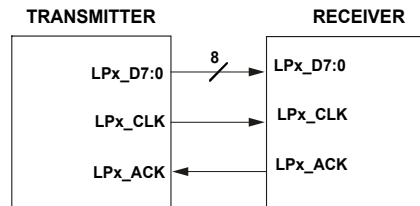


Figure 14-1: Link Port Block Diagram

External Connections

As shown in the *Link Port Pin Connections* figure, a link port has eight data lines (LP_D0 – LP_D7), a clock line (LP_CLK), and an acknowledge line (LP_ACK). A link port can act as either a transmitter or a receiver but not both at the same time.



X DENOTES THE LINK PORT NUMBER, 0-1

Figure 14-2: Link Port Pin Connections

Use external pull-downs for the LP_CLK and LP_ACK pins so that the link port can enable the transmitter and receiver, irrespective of the state of the other.

Internal Blocks

As shown in the block diagram, the link ports have independent modules for transmit and receive. If enabled as transmitter, the link port uses a 2 deep 32-bit FIFO. If enabled as a receiver, the port uses a 4 deep 32-bit FIFO. The core MMR access bus interfaces with these FIFOs. The distributed DMA engines (DDE) use the system cross bar (SCB) interface to access the FIFO. The link port uses the LP_CTL.TRAN bit to determine whether the module is enabled for transmit or receive operation.

Architectural Concepts

The following sections describe the architectural concepts of the link port.

- [Link Port Protocol](#)
- [FIFO Buffers](#)
- [Handshake for Link Port Enable Process](#)
- [Clocking](#)
- [Multi-Processor Connectivity](#)

Link Port Protocol

A link port transmitted word consists of 4 bytes and the communication proceeds as follows.

1. The transmitter asserts the link port clock (LP_CLK) with each byte of data. The receiver uses the falling edge of LP_CLK driven by the transmitter to latch the byte.
2. When the receiver is ready to accept another word in the receive buffer it asserts the acknowledge signal, LP_ACK.

3. The transmitter samples LP_ACK driven by the receiver at the beginning of each word transmission. If LP_ACK is deasserted, then the transmitter does not transmit the next word.
4. The transmitter leaves LP_CLK high and continues to drive the first byte of the next word until LP_ACK is asserted.
5. When this assertion occurs, the transmitter drives LP_CLK low. The transmission of the next word starts. If the transmit buffer is empty, LP_CLK remains low until the buffer refills, regardless of the state of LP_ACK.

The LP_ACK signal can deassert when it anticipates that the buffer could fill. The receiver reasserts the LP_ACK signal as soon as the internal DMA grant signal has occurred or the core reads the receive buffer. Either of these actions frees a buffer location.

NOTE: The LP_ACK signal inhibits transmission of the next word and not of the current byte.

The LP_ACK signal provides a handshake between the receiver and transmitter in the following configurations.

- When configured as a transmitter, the port drives both the data and the clock while LP_ACK is three-stated. In this mode, LP_CLK is always synchronous with OCLK0_0.
- When configured as a receiver, the link port drives the acknowledge signal and the data and clock lines are three-stated. In this case, the external LP_CLK signal can either be synchronous or asynchronous with OCLK0_0.
- When the link port is disabled, the data, clock, and acknowledge signals are three-stated.

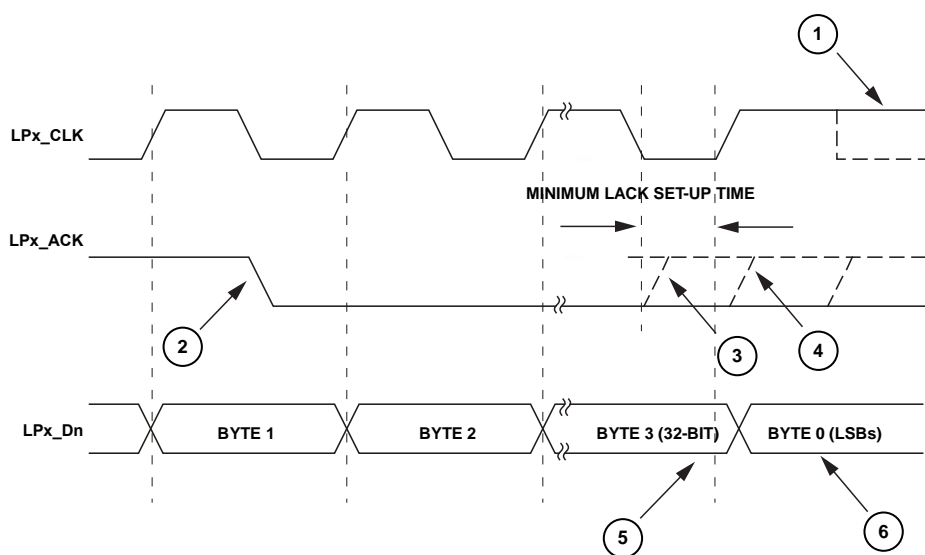


Figure 14-3: Link Port Communication and Handshake Waveform

The following list describes the stages shown in the *Link Port Communication and Handshake Waveform* figure.

1. LP_CLK stays high at byte 0 when LP_ACK is sampled low on the previous LP_CLK rising edge. LP_CLK high indicates a stall.
2. The LP_ACK signal can deassert after byte 0.

3. The LP_ACK signal reasserts as soon as the link buffer is not full (depending on Rx FIFO conditions).
4. The transmitter samples LP_ACK to determine whether to transmit the next word.
5. The receiver accepts the remaining word even if LP_ACK is deasserted. The transmitter does not send the following word.
6. Transmission of data for next word is held until LP_ACK is asserted.

The transmitter samples the LP_ACK signal. If the signal is high, the transmitter gives out the falling edges of LP_CLK for data sampling. The LP_ACK signal is first sampled at the rising edge of OCLK0_0. One more OCLK0_0 stage synchronizes the signal further. This synchronized signal is given to the subsequent logic. The LP_CLK falling edge is aligned with OCLK0_0 falling edge in a 1:1 clock ratio mode and with the OCLK0_0 rising edge for the rest of the clock ratios. The following figures explain how the synchronization is maintained between the LP_ACK and LP_CLK signals.

In the following figure, synchronizing time is guaranteed to be 1.5 OCLK0_0 cycles.

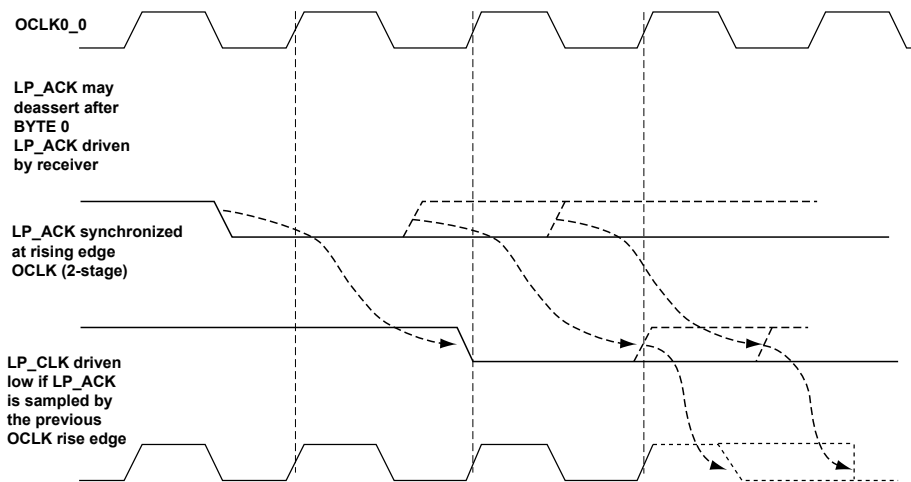


Figure 14-4: LP_ACK Synchronization for OCLK0_0:LP_CLK=1:1

In the following figure, synchronizing time is guaranteed to be 2 OCLK0_0 cycles.

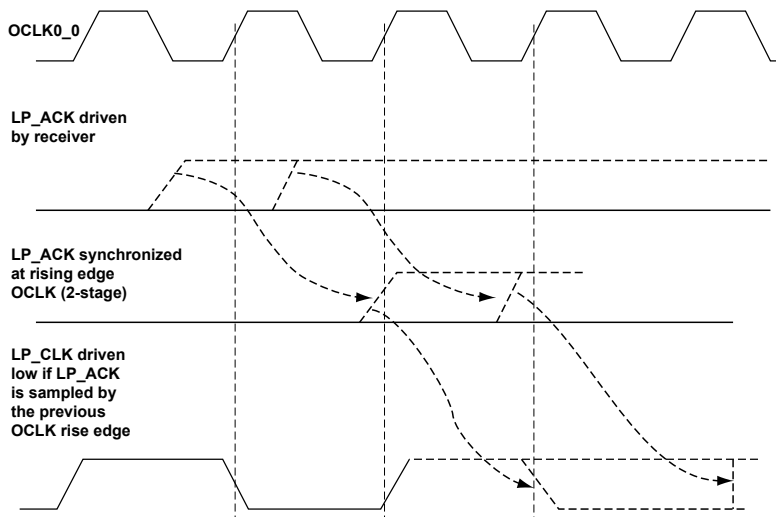


Figure 14-5: LP_ACK Synchronization for OCLK0_0: LP_CLK=1:2, 1:4 and Up

The link port uses the value programmed in the LP_DIV register at the transmitter to determine the frequency of the link port clock (LP_CLK). However, the signal appearing on the LP_CLK pin is also dependent on the status of the LP_ACK pin driven by the receiver. The *Relationship Between Internal Link Port Clock and Link Port Clock at the Pins* figure shows this relationship.

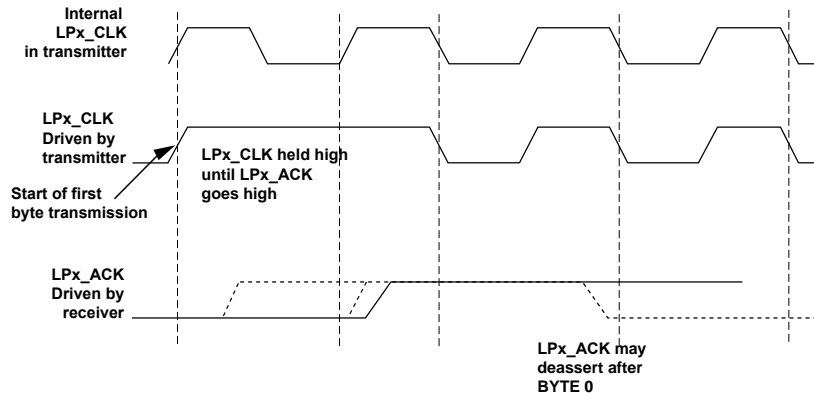


Figure 14-6: Relationship Between Internal Link Port Clock and Link Port Clock at the Pins

FIFO Buffers

When a link port is configured as transmitter, the link port uses a 2-deep FIFO buffer. A shift register unpacks the single 32-bit word to four 8-bit data bytes. As the FIFO has space for more data, the link port makes a new DMA request. If the FIFO becomes empty, the LP_CLK signal is deasserted. The core can access FIFO through the LP_TX register.

The core or DMA makes three writes (2-stage FIFO and 1 shift register) to the transmit buffer before it signals a full condition. The link port uses the LP_STAT.FFST bit field to reflect the status of the FIFO but not the shift register full or empty condition. However, the program can poll the LP_STAT.LPBS bit to discover whether the link

port is driving data from the shift register to the pins. The `LP_STAT.LPBS` bit is also set when receiver has held off transmission by driving `LP_ACK` low.

NOTE: When the 2-deep FIFO and the output shift-register overflow, any further write to the link port buffer overwrites the input stage of the FIFO.

NOTE: The core can also read the transmit FIFO through the data `LP_TX` register.

NOTE: If the transmitter is disabled while performing writes to the transmit FIFO, a FIFO full condition is signaled after two writes.

The transmit buffer registers have shadow registers. Using these shadow registers, both stages of the 2-deep FIFO can be read without updating the status registers. The `LP_TXIN_SHDW` register corresponds to the input stage of the FIFO. The `LP_TXOUT_SHDW` register corresponds to the output stage of the FIFO as shown in the *Transmit FIFO path* figure.

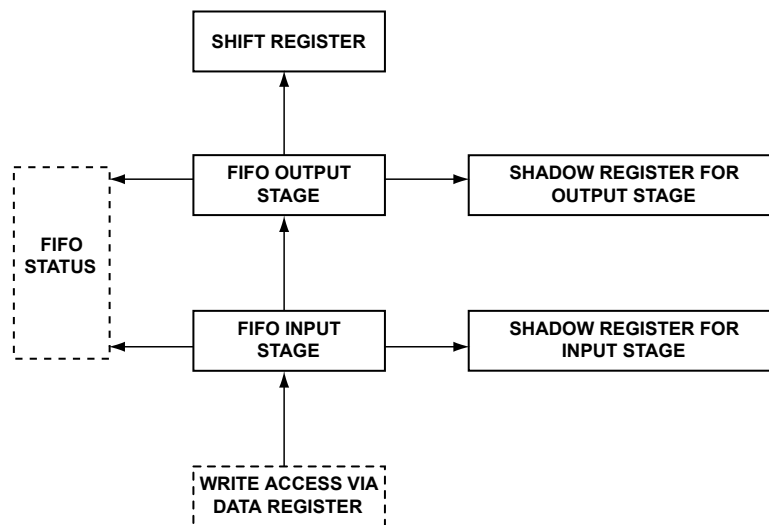


Figure 14-7: Transmit FIFO path

When a link port is configured as receiver, data transfers to the core or DMA from the full 4-deep receive FIFO. An internal packing register packs data to 32 bits. Four reads can occur from the receive buffer by the core or DMA before it signals an empty condition. The link port uses the `LP_STAT.FFST` bits to reflect the status of the 4-deep read buffer FIFO. The core can access this FIFO through the `LP_RX` register.

NOTE: When receive FIFO overflows (`LP_STAT.ROVF` bit=1), any further data from the transmitter is lost. Only the data retained in the receive FIFO can be retrieved further.

The receiver drives the `LP_ACK` output signal low, after the first byte of data for the next-to-last empty slot (in the 4-deep FIFO) is received. This functionality prevents data loss due to the transmitter starting transmission of the next word before the `LP_ACK` signal reaches the transmitter. (The timing is due to the larger delay in synchronization.) This functionality guarantees that even after allowing for the extra synchronization cycle in the transmitter and receiver, there is no overflow in the receive FIFO. The *LACK Generation Based on Receive FIFO Status* figure

shows how FIFO slots influence the acknowledge signal generation. The grayed sections show received data. The white sections show empty locations where the decision to pull LP_ACK high is taken.

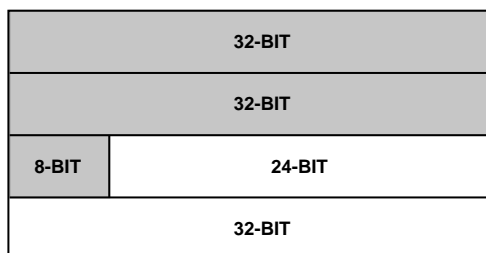


Figure 14-8: LACK Generation Based on Receive FIFO Status

NOTE: The link port uses a 4-deep receive FIFO only under a worst case situation, as mentioned. In all other cases, respond as if the FIFO has only a 3-deep stage. The LP_ACK signal is pulled high before the last stage of the FIFO.

The link port has memory-mapped buffers for both receive and transmit operations. A JTAG-based emulator can read the FIFO which can cause unexpected problems in data transfers. This activity can only happen during an emulation event (typically hitting a breakpoint or single-stepping). The emulator issues core reads through JTAG. To work around this issue, see the tools documentation for more information.

Handshake for Link Port Enable Process

In a link port-based system, the transmitter and the receiver can be enabled at different times. Use external pull-downs for the LP_CLK and LP_ACK signals.

If the receiver is enabled before the transmitter, the external pull-down holds the LP_CLK signal of the transmitter low. The receiver is held off. The receiver can wait for a rising edge on the LP_CLK signal to assert its receive service request interrupt. This rising edge occurs only when transmitter starts driving the first data on to the bus, after the application enables it.

If the transmitter is enabled before the receiver, the external pull-down holds the LP_ACK signal of the receiver low. Transmission is held off. Refer to the *Enable the Transmitter Before the Receiver* figure. The transmitter can wait for a rising edge on the LP_ACK signal to assert its transmit service request interrupt. This rising edge is asserted as soon as the receiver is enabled after the hardware drives the LP_ACK high.

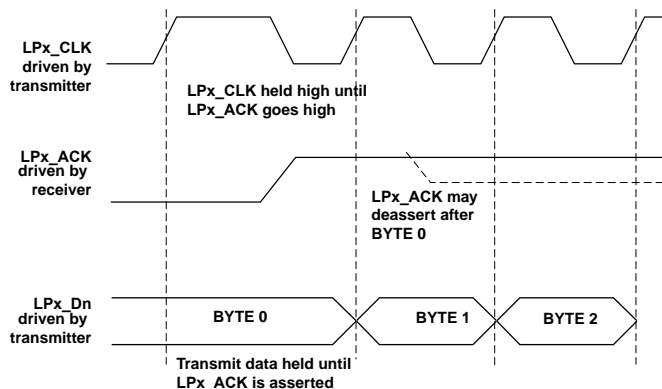


Figure 14-9: Enable the Transmitter Before the Receiver

NOTE: Service request interrupts or status are asserted only when the link port (receiver or transmitter) is disabled.

Clocking

The link port clock (LP_CLK) is derived from the internal system clock (OCLK0_0). The link port clock to system clock ratio can be configured in the LP_DIV register. This value applies to the transmitter only. The receiver can operate at any asynchronous frequency up to the maximum frequency, independent of the ratio programmed. The following formula describes the relationship between the frequency of the link port clock, the OCLK0_0 frequency, and the LP_DIV value.

- $f_{LP_CLK} = f_{OCLK0_0}$ or $f_{LP_CLK-MAX}$ if $DIV = 0$
- $f_{LP_CLK} = f_{OCLK0_0} / (2 \times DIV)$ if $DIV > 0$

Where: f_{LP_CLK} = link clock frequency, $f_{LP_CLK-MAX}$ = link clock maximum frequency, and f_{OCLK0_0} = system clock frequency.

While programming the LP_DIV register to select the clock ratio, ensure that the LP_CLK frequency does not exceed the maximum frequency supported for the device. For supported frequencies, see the product-specific data sheet.

NOTE: The Drive Strength (DS) value must be changed to b(010) for the LPCLK pin to operate the link port at the maximum frequency of 125 MHz. Refer the PADS_PORTB0_DS and PADS_PORTC0_DS register bit descriptions for details.

Multi-Processor Connectivity

Link ports can operate independently, allowing glueless connection with external processors. Link ports have dedicated DMA channels, allowing independent data transfers. The following group of figures shows some examples of different bus connection topology that can be used in multi-processor system design. The inter-connection methods are not limited to these examples.

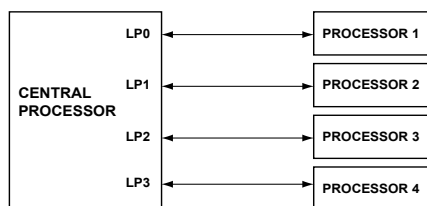


Figure 14-10: Central Processor-Based Model

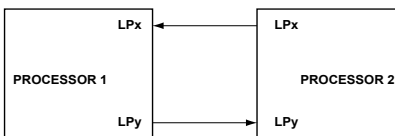


Figure 14-11: Link Port Full-Duplex Transfer Model

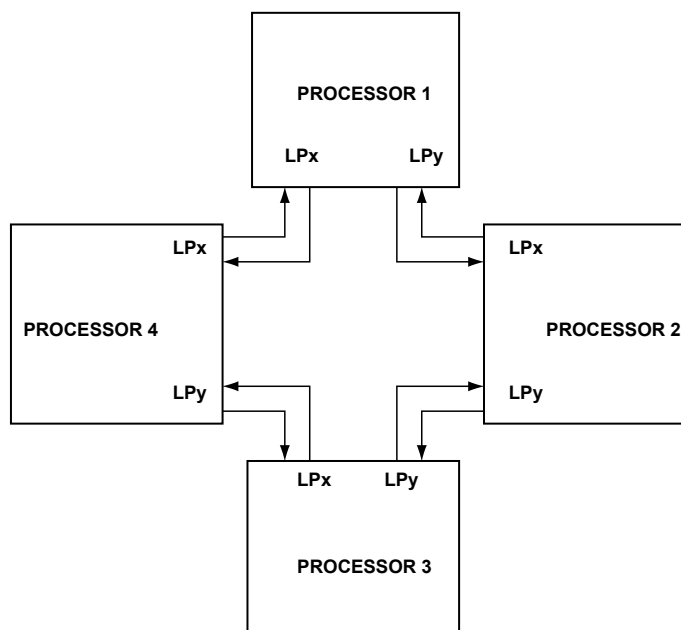


Figure 14-12: Link Port Ring Model

The link port protocol does not include built-in support for multiple masters. However, there can be situations where multiple devices try to become the bus master at the same time. Multi-master conflicts can be resolved using token passing. In token passing, the token is a software flag that passes between processors.

At reset, the token is set to reside in the link port of one device, making it the master and the transmitter. When a receiver (slave) wants to become the master, it can assert its `LP_ACK` signal to get the attention of the master. The master knows, through the software protocol, whether to respond with actual data or whether the token is requested. If the master wishes to give up the token, it can send back a user-defined token release word and thereafter clear its token flag.

Simultaneously, the slave sets its token and can thereafter transmit. The token release word can be any user-defined value. Because the transmitter and receiver expect a code word, this word does not need to be exclusive of normal

data transmission. If the master wishes to give up the token, it can send back a user-defined token release word and thereafter clear its token flag. Simultaneously, the slave examines the data sent back and if it is the token release word, the slave sets its token, and can thereafter transmit.

The link port protocol includes handshake mechanism to inform the other end of transfer (transmit or receive) of an enable instance. However, it does not support handshakes to inform a disable instance, while a chunk of data transfers. The application must assume the disabled state of the other end, and take appropriate action.

For example, in a multi-processing environment, a receiver did not read its full FIFO for an extended time due to internal bus arbitrations. The transmitter can require software or a peripheral timer-based timeout to inform the application that the `LP_ACK` signal is low for an extended time period.

LP Operating Modes

The link port does not have particular modes of operation, as the peripheral is based on a simple protocol. The following sections explain the data transfer modes, using the core and using DMA.

- [Core Data Transfers](#)
- [DMA Data Transfers](#)

LP Data Transfer Modes

This section describes link port DMA and core data transfers.

Core Data Transfers

If DMA is disabled for a link port buffer, the processor core can write or read internal FIFO buffers as a memory-mapped register through the MMR access bus. In order to avoid FIFO overflow or underflow, the core can access the FIFO registers in one of the two following ways.

1. Access link port registers using an interrupt service routine (ISR) mapped to the data request interrupt of the link port. The interrupt request remains high only if the FIFO is accessible (if the FIFO is not full in transmit mode and not empty in receive mode).
2. Poll the FIFO status bits of the `LP_STAT` register. Write to the transmit FIFO if not full or read from the receive FIFO if not empty.

DMA Data Transfers

Dedicated DMA channels are available for each link port. DMA-related activity is explained in the following steps.

1. Data Receive – Once the DMA channel and link port module are configured and enabled, the external device begins writing data to the FIFO through the data pins of the link port. The FIFO detects this activity and in turn sends a DMA request. After the request is granted, the DMA transfer progresses until the FIFO is empty.
2. Data Transmit – Once the DMA channel and link port module are configured and enabled, setting the `LP_CTL.EN` bit automatically asserts a DMA request when the transmit FIFO is empty. After the request is granted, DMA fills the FIFO. The external device begins reading data from the FIFO through the data pins of

the link port. The FIFO detects that there is room in the buffer and asserts another DMA request, continuing the process.

LP Event Control

This section describes how the link port uses interrupts and status signals.

Interrupt Signals

Each link port has two dedicated interrupt lines registered with the system event controller—a data request interrupt and a status interrupt. Data request interrupts are asserted based on FIFO conditions for data transfer. Status interrupts are asserted when a service request status or an overflow status is set. The following list explains each of these interrupts.

- **Data Request Interrupt.** Asserted if the FIFO is not full in transmission mode and the FIFO is not empty in reception mode. This functionality serves as a core triggered interrupt in non-DMA mode and as the DMA interrupt request in DMA mode. Generation of this interrupt is based on the `LP_STAT.FFST` (status bit of the link port buffer).
- **Link Port Transmit Service Request Interrupt (LTRQ).** Allow a disabled link port to generate an interrupt when an external access is attempted. When a link port is configured as transmitter, the transmit service request interrupt is enabled by setting the `LP_CTL.TRQMSK` bit. When set, an external receiver can indicate to the disabled transmitter that it must receive data through the connected link port. The receiver does so by driving a high level on the `LP_ACK` line. When the `LP_ACK` of the disabled transmitter link port is detected high, a `LP_STAT.LTRQ` interrupt is generated. The transmitter can enable itself for data transfer with the receiver. The link port needs a pull-down on `LP_ACK` for this feature to function properly.
- **Link Port Receive Service Request Interrupt (LRRQ).** When a link port is configured as receiver, this interrupt is enabled by setting the `LP_CTL.RRQMSK` bit. When set, an external transmitter can indicate to the disabled receiver that it must receive data through the connected link port. The transmitter does so by driving out the first data. When the `LP_CLK` of the disabled receiver link port is detected high, a `LP_STAT.LRRQ` interrupt is generated. The receiver can further enable itself for data transfer with the transmitter. The link port needs a pull-down on the `LP_CLK` signal for this feature to function properly.
- **Link Port Receive Overflow Interrupt (LPOVF).** Generated when the receiver FIFO overflows and is enabled by setting the `LP_CTL.ROVFMSK` bit. This interrupt can happen if the transmitter continues to transmit data even though the receiver has deasserted `LP_ACK` signal causing the receive FIFO to overflow.

Enabling Link Port Interrupts

A data request interrupt is fed to the system event controller directly and can be controlled separately from the application.

To mask the interrupt, set the mask bits in `LP_CTL` register corresponding to service interrupts and the overflow interrupt. These interrupts are OR'ed and fed to the SIC as a single `LP_STAT` interrupt. These interrupts are latched and stored in the associated bits of `LP_STAT` register. If an `LP_STAT` interrupt occurs, in the ISR, programs

can read the `LP_STAT` register bits to determine the type of interrupt. These bits are write-one-to-clear (W1C); writing one to the bit resets the bit and disables the corresponding interrupt.

Status and Error Signals

This section explains the various status signals in the `LP_STAT` register.

- *Transfer Status signals.* The link port uses the bus status bit (`LP_STAT.LPBS`) to give the status of the bus condition (busy or idle), when the link port is configured as transmitter. The `LP_STAT.LPBS` is high if the link port drives data into the link port pins. Programs can poll this bit after polling the `LP_STAT.FFST` bit to disable the link port safely.

The link buffer status (`LP_STAT.FFST`) field directly indicates the status of the FIFO (including empty or full conditions) during data transfer. Software can poll this field in the `LP_STAT` register before writing to the FIFO (in case of transmission) or reading from the FIFO (in case of reception). The `LP_STAT.FFST` bit is automatically cleared when the link port is disabled.

- *Transfer Request Status signals.* The link port uses the receive request status (`LP_STAT.LRRQ`) bit to indicate that an external receiver wants to receive data (in case the link port is a disabled transmitter). The link port uses the transmit request status (`LP_STAT.LTRQ`) bit to indicate that an external transmitter wants to send data (in case the link port is a disabled receiver). Software can poll these bits to enable the transmitter or receiver accordingly.
- *Error Status signals.* In receive mode 32-bit data is received in four chunks of 8-bit data. This data is then packed to a single 32-bit data before loading the FIFO. The link buffer error status (`LP_STAT.LPACK`) bit is high during this packing process and goes low after packing.

The link port overflow status (`LP_STAT.ROVF`) bit is set when the receive FIFO overflows. This event can occur if the transmitter continues to transmit data even though the receiver has deasserted `LP_ACK` causing the receiver FIFO to overflow.

LP Programming Model

The following sections provide information on configuring the operating mode and enabling the link ports.

- [Setting Up a DMA Transmit Operation](#)
- [Setting Up a DMA Receive Operation](#)
- [Setting Up a Core Transmit Operation](#)
- [Setting Up a Core Receive Operation](#)

Setting Up a DMA Transmit Operation

This following procedure describes the typical steps for configuring the link ports in DMA transmit mode.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.

2. Configure the drive strength value to b(010) for the link port clock pins to operate the link port at maximum frequency (`PADS_PORTB0_DS` and `PADS_PORTC0_DS` registers).
3. Install interrupt handlers for DMA and for transfer status (service request interrupt).
4. Configure the link port to transmit by setting the `LP_CTL` bit and enable the transmit request interrupt mask by setting the `LP_CTL.TRQMSK` bit.
5. Program the link port clock divider by writing a value to the `LP_DIV` register.
6. If using DMA stop mode or auto buffer mode, program the appropriate DMA registers.
ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Stop/Auto, `DMA_CFG.PSIZE = 1`, `DMA_CFG.MSIZE = 4`, interrupt generation and memory read).
7. Wait for the link port receiver (connected externally) to be enabled. The application can wait for the transmit service request interrupt to assert.
8. Clear the transmit service request interrupt status by writing 1 to the `LP_STAT.LTRQ` bit.
9. Enable DMA by setting the `DMA_CFG.EN` bit.
10. Enable the link port by setting the `LP_CTL.EN` bit.
11. Wait for DMA to assert a transfer completion interrupt.
12. Clear the DMA interrupt source by writing 1 to the `DMA_STAT.IRQDONE` bit.

Setting Up a DMA Receive Operation

This section describes the typical steps for using the link ports in DMA receive mode.

1. Enable the link port pins in GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Install interrupt handlers for DMA and for transfer status (service request interrupt).
3. Configure the link port for reception (clear the `LP_CTL.TRAN` bit) and enable the receive request interrupt mask by setting the `LP_CTL.RRQMSK` bit.
4. If using DMA stop mode or auto buffer mode, program the DMA registers.

ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Stop/Auto, `DMA_CFG.PSIZE = 1`, `DMA_CFG.MSIZE = 4`, interrupt generation and memory write).

5. If using DMA array mode or list mode, create DMA configuration data structures filled with components.

ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Array/List, `DMA_CFG.PSIZE = 1`, `DMA_CFG.MSIZE = 4`, interrupt generation, memory write and fetch =4/5) and `DMA_DSCPTR_NXT` register (if list mode). Further, program

DMA configuration register (Array/List, `DMA_CFG.PSIZE = 1`, `DMA_CFG.MSIZE = 4`, Memory Write and Fetch `=4/5`) and program the `DMA_DSCPTR_NXT` register (if list mode).

6. Wait for the link port transmitter (connected externally) to be enabled with subsequent data transmission. The application can wait for the receive service request interrupt to assert.
7. Clear the receive service request interrupt status by writing 1 to the `LP_STAT.LRRQ` bit.
8. Enable DMA by setting the `DMA_CFG.EN` bit.
9. Enable the link port by setting the `LP_CTL.EN` bit.
10. Wait for DMA to assert the transfer complete interrupt.
11. Clear the DMA interrupt source by writing 1 to the `DMA_STAT.IRQDONE` bit of the DMA status register.

Setting Up a Core Transmit Operation

This section describes the typical steps for using the link ports in processor core based transmission.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Configure the drive strength value to b(010) for the link port clock pins to operate the link port at maximum frequency (`PADS_PORTB0_DS` and `PADS_PORTC0_DS` registers).
3. Install interrupt handlers for data transfer and for transfer status (service request interrupt). The interrupt handlers for data transfer are the same source or ID as the DMA interrupt line in the SEC.
4. Configure the link port for transmission by setting the `LP_CTL.TRAN` bit) and enable the transmit request interrupt mask by setting the `LP_CTL.TRQMSK` bit).
5. Program the link port clock divider by writing a value in to the `LP_DIV` register.
6. Wait for the link port receiver (connected externally) to be enabled. The application can wait for a transmit service request interrupt to assert.
7. Clear the transmit service request interrupt status by writing 1 to the `LP_STAT.LTRQ` bit.
8. Enable the link port by setting the `LP_CTL.EN` bit.
9. The data request interrupt is asserted whenever there is free space in the FIFO. The application can write to the `LP_TX` register based on the FIFO conditions (half or empty) reflected in the `LP_STAT.FFST` bit field.

Setting Up a Core Receive Operation

This section describes the typical steps for using the link ports in processor core-based reception.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Install interrupt handlers for data transfer and for transfer status (service request interrupt). The interrupt handlers for data transfer are the same source or ID as the DMA interrupt line in the SEC).

3. Configure link port for reception (clear `LP_CTL.TRAN` bit). Enable the receive request interrupt mask bit (set `LP_CTL.RRQMSK`).
4. Wait for the link port transmit (connected externally) to be enabled with subsequent transmission of data. The application can wait for receive service request interrupt to be asserted.
5. Clear the receive service request interrupt status by writing 1 to the `LP_STAT.LRRQ` bit.
6. Enable the link port by setting the `LP_CTL.EN` bit.
7. The data request interrupt is asserted whenever there is free space in the FIFO. The application can read from the `LP_RX` register based on the FIFO conditions (1 or 2 or 3 data available) which is reflected in the `LP_STAT.FFST` bit field.

ADSP-2156x LP Register Descriptions

Link Port (LP) contains the following registers.

Table 14-6: ADSP-2156x LP Register List

Name	Description
<code>LP_CTL</code>	Control Register
<code>LP_DIV</code>	Clock Divider Value Register
<code>LP_RX</code>	Receive Buffer Register
<code>LP_STAT</code>	Status Register
<code>LP_TX</code>	Transmit Buffer Register
<code>LP_TXIN_SHDW</code>	Shadow Input Transmit Buffer Register
<code>LP_TXOUT_SHDW</code>	Shadow Output Transmit Buffer Register

Control Register

The LP_CTL register provides LP interrupt masking, selection of transfer direction, and link port enable.

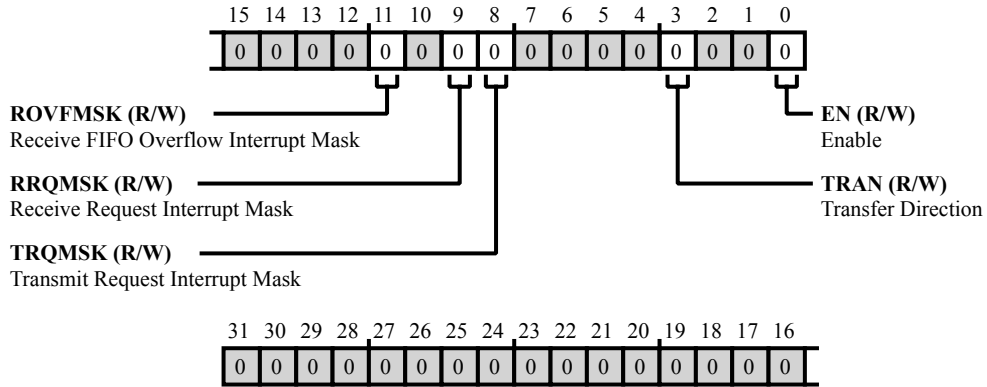


Figure 14-13: LP_CTL Register Diagram

Table 14-7: LP_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	ROVFMSK	Receive FIFO Overflow Interrupt Mask. Receive FIFO Overflow Interrupt Mask
		0 Mask Disable Receive FIFO Overflow Interrupt
		1 Unmask Enable Receive FIFO Overflow Interrupt
9 (R/W)	RRQMSK	Receive Request Interrupt Mask. Link Port Receive Request Mask
		0 Mask Disable Receive Request interrupt.
		1 Unmask Enable Receive Request interrupt.
8 (R/W)	TRQMSK	Transmit Request Interrupt Mask. Link Port Transmit Request Mask
		0 Mask Disable Transmit Request interrupt.
		1 Unmask Enable Transmit Request interrupt.
3 (R/W)	TRAN	Transfer Direction. The LP_CTL . TRAN bit selects the transfer direction as transmit (if set) or receive (if cleared) for link buffer.
		0 Receive Direction transfer is receive
		1 Transmit Direction transfer is transmit

Table 14-7: LP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The LP_CTL.EN enables or disables the link port. When the processor disables the port (LP_CTL.EN transitions from high to low), the processor clears the corresponding LP_STAT bits.	
		0	Disable linkport
		1	Enable linkport

Clock Divider Value Register

The `LP_DIV` register selects the divisor for ratio between the internal LP clock (LCLK) and system clock (SCLK). This programming is applicable only for the transmitter. The receiver can operate at any asynchronous frequency up to the maximum frequency independent of the ratio programmed.

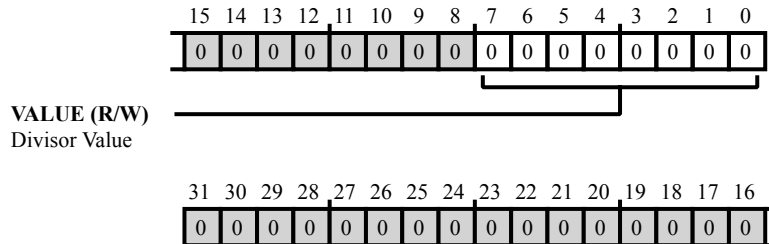


Figure 14-14: LP_DIV Register Diagram

Table 14-8: LP_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	<p>Divisor Value.</p> <p>The <code>LP_DIV.VALUE</code> bits select the clock divider (relating the LP' internally generated clock (LCLK) to the system clock (SCLK). The <code>LP_DIV.VALUE</code> should be programmed prior to LP enable.</p> <p>For <code>LP_DIV.VALUE = 0</code>, <code>LCLK = SCLK</code></p> <p>For <code>LP_DIV.VALUE = xxxxxxxx</code>, <code>LCLK = SCLK / (2 x DIV)</code></p>

Receive Buffer Register

The `LP_RX` register buffers the receive data flow through the LP. The receive buffer is a four-location deep FIFO. In the receive buffer, data is transferred to the core or DMA from the receive FIFO where an internal register does the packing. This packing register is not software accessible. For more information on LP buffer features and operations, see the LP functional description.

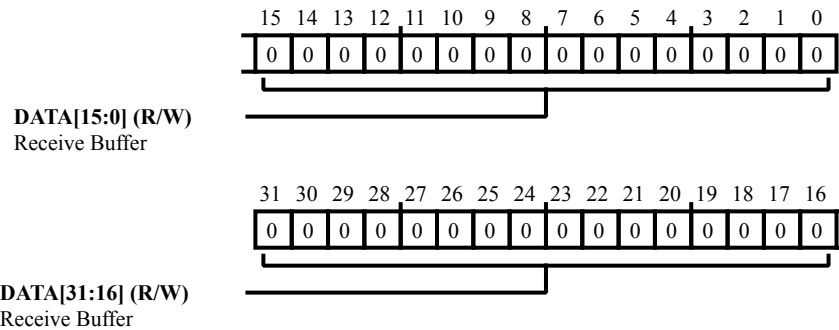


Figure 14-15: LP_RX Register Diagram

Table 14-9: LP_RX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Receive Buffer.

Status Register

The `LP_STAT` register provides status information on link port interrupts, FIFO, buses, and receive/transmit requests.

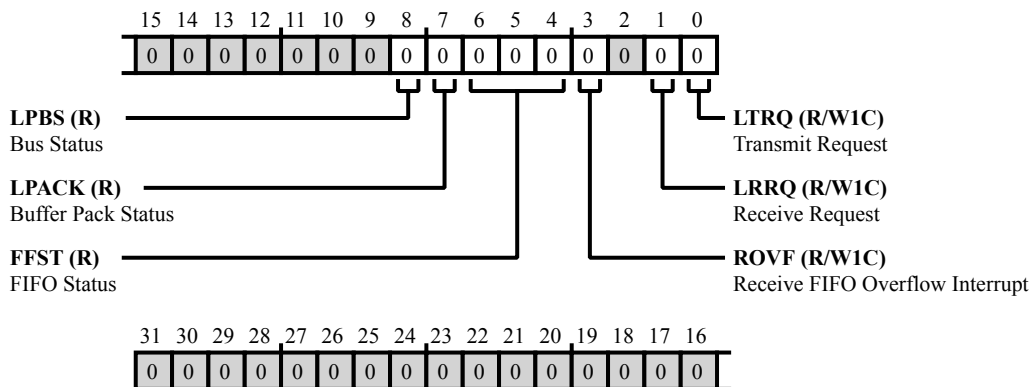


Figure 14-16: LP_STAT Register Diagram

Table 14-10: LP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/NW)	LPBS	Bus Status. The <code>LP_STAT.LPBS</code> bit indicates the LPDAT bus status. <code>LP_STAT.LPBS</code> is kept high if data is being driven by the link port into the <code>LP_D[n]</code> pins.
		0 Bus is Idle Link Port Bus is idle
		1 Bus Busy Link Port Bus is busy
7 (R/NW)	LPACK	Buffer Pack Status. The <code>LP_STAT.LPACK</code> bit indicates packing status. In receive mode, 32-bit data is received in 4 blocks of 8-bit data. Then, the data is packed to get a single 32-bit data before loading the FIFO. The <code>LP_STAT.LPACK</code> bit is high during this packing process and goes low after packing. In transmit mode, 32-bit data in the FIFO is unpacked to 4 blocks of 8-bit data before sending. The <code>LP_STAT.LPACK</code> is high during unpacking.
		0 Packing Complete Packing done
		1 Packing Incomplete Packing is in progress
6:4 (R/NW)	FFST	FIFO Status. The <code>LP_STAT.FFST</code> bits indicate the FIFO status. These bits are cleared when the LP is disabled.
		0 TX - Empty; RX - Empty Link buffer (TX OR RX) empty

Table 14-10: LP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		1 TX - Reserved ; RX - Has 1 data word RX has 1 word of data. TX reserved
		2 TX - Reserved; RX - Has 2 data words RX has 2 word of data. TX reserved.
		3 TX - Reserved; RX - Has 3 data words RX has 3 word of data. TX reserved.
		4 TX - One Word; RX -Has 4 data words RX has 4 word of data. TX 1 word of data.
		5 Reserved
		6 TX - FIFO Full; RX - Reserved RX reserved. TX reserved.
		7 Reserved
3 (R/W1C)	ROVF	Receive FIFO Overflow Interrupt. This interrupt is generated when the receiver FIFO overflows. This overflow may happen if the transmitter continues to transmit data even though the receiver has de-asserted the LP_ACK pin.
1 (R/W1C)	LRRQ	Receive Request. The LP generates this interrupt when the LP_CLK pin of a disabled link port (the receiver) is forced high by another link port (the transmitter).
0 (R/W1C)	LTRQ	Transmit Request. The LP generates this interrupt when the LP_ACK pin of a disabled link port (the transmitter) is forced high by another link port (the receiver).

Transmit Buffer Register

The `LP_TX` register buffers the transmit data flow through the LP. The transmit buffer is two words deep. In the transmit buffer, the input stage of the FIFO is used to accept core data or DMA data from internal memory, and the data is transferred to the link port interface from the output stage of the FIFO. The output stage performs the unpacking in the transmit buffer. The least significant byte is transmitted first. As each word is unpacked and transmitted, the next location in FIFO becomes available and a new DMA request is made if DMA is enabled. If the register becomes empty, the LP asserts the `LP_CLK` signal. For more information on LP buffer features and operations, see the LP functional description.

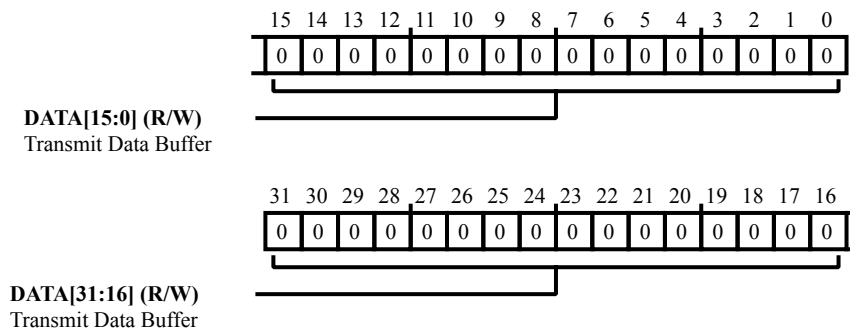


Figure 14-17: LP_TX Register Diagram

Table 14-11: LP_TX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit Data Buffer.

Shadow Input Transmit Buffer Register

The `LP_TXIN_SHDW` register contains the same data as the input stage of the transmit buffer. Read of this shadow transmit buffer does not update the `LP_STAT` register.

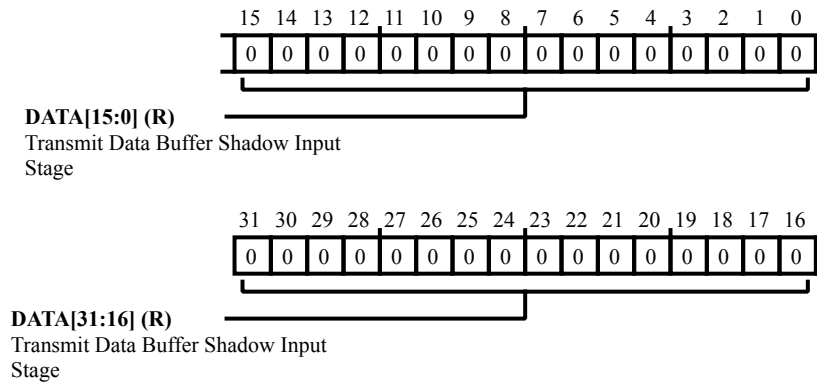


Figure 14-18: LP_TXIN_SHDW Register Diagram

Table 14-12: LP_TXIN_SHDW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Transmit Data Buffer Shadow Input Stage.

Shadow Output Transmit Buffer Register

The `LP_TXOUT_SHDW` register contains the same data as the output stage of the transmit buffer. Read of this shadow transmit buffer does not update the `LP_STAT` register.

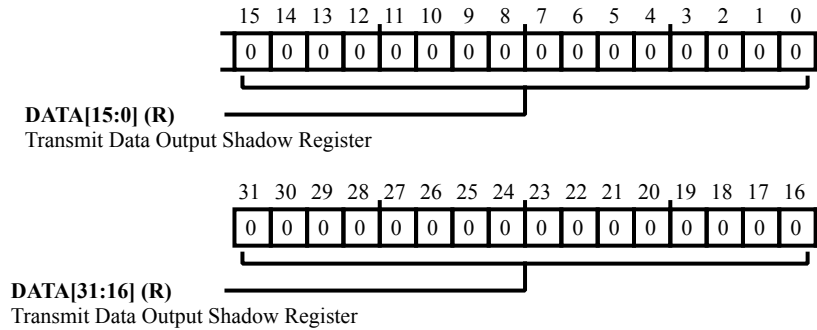


Figure 14-19: LP_TXOUT_SHDW Register Diagram

Table 14-13: LP_TXOUT_SHDW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Transmit Data Output Shadow Register.

15 Serial Peripheral Interface (SPI)

The serial peripheral interface is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

With a range of configurable options, the SPI ports provide a glueless hardware interface with other SPI-compatible devices in master mode, slave mode, and multimaster environments. The SPI peripheral includes programmable baud rates, clock phase, and clock polarity. The peripheral can operate in a multimaster environment by interfacing with several other devices, acting as either a master device or a slave device. In a multimaster environment, the SPI peripheral uses open-drain outputs to avoid data bus contention. The flow control features enable slow slave devices to interface with fast master devices by providing an SPI ready pin which flexibly controls the transfers.

NOTE: SPI peripherals SPI0, SPI1 and SPI2 operate in the SCLK0 domain. For details on SCLK programming, refer the *Clock Generation Unit (CGU)* chapter.

SPI Features

The SPI module supports the following features:

- Full-duplex, synchronous serial interface
- 8, 16-bit and 32-bit word sizes
- Programmable baud rate, clock phase, and polarity
- Programmable interframe latency
- Flow control
- Support for Fast and DIOM modes
- SPI1 and SPI2 support quad mode. Memory-mapped mode is supported by SPI2 only.
- Independent receive and transmit DMA channels

- Burst transfer mode for non-DMA write accesses

SPI Functional Description

This section provides information on the function of the SPI module.

Shift register functionality

The SPI is essentially a shift register that serially transmits and receives data bits to or from other SPI devices. During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line synchronizes shifting and sampling of the information on the two serial data lines.

Master slave functionality

During a data transfer, one SPI system acts as the link master which controls the data flow. The other system acts as the slave, which has data shifted into and out of it by the master. Different devices can take turn being masters, and one master can simultaneously shift data into multiple slaves (broadcast mode). However, only one slave can drive its output to write data back to the master at any given time. This rule must be enforced in the broadcast mode. Several slaves can be selected to receive data from the master in this mode. But only one slave can be enabled to send data back to the master.

Enhanced operating modes

SPI supports enhanced modes of operation like fast mode, DIOM, and Quad-SPI, and optional flow control. In fast mode, received data is sampled on the transmit edge instead of the standard receive edge, thus enabling a full-cycle path for the received data. In DIOM, both MOSI and MISO are configured as input or output pins, and 2 bits are shifted in or out on each receive or transmit edge. In Quad-SPI mode, SPI_D3:0 are configured as input or output pins and 4 bits are shifted in or out on each receive or transmit edge. A slower slave can use flow control to stall a faster master device.

Single and multi-master use

The SPI can be used in a single master as well as multi-master environment. The SPI_MOSI, SPI_MISO, and the SPI_CLK signals are all tied together in both configurations. SPI transmission and reception can be enabled simultaneously or individually, depending on SPI_RXCTL and SPI_TXCTL settings. In broadcast mode, several slaves can be enabled to receive, but only one slave must be in transmit mode and driving the SPI_MISO line.

ADSP-2156x SPI Register List

The Serial Peripheral Interface (SPI) provides a full-duplex, synchronous serial interface, which supports both master/slave modes and multi-master environments. The SPI's baud rate and clock phase/polarities are programmable, and it has integrated DMA channels for both transmit and receive data streams. A set of registers governs SPI operations. For more information on SPI functionality, see the SPI register descriptions.

Table 15-1: ADSP-2156x SPI Register List

Name	Description
SPI_CLK	Clock Rate Register
SPI_CTL	Control Register
SPI_DLY	Delay Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

ADSP-2156x SPI Interrupt List

Table 15-2: ADSP-2156x SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
69	SPI2_TXDMA	SPI2 TX DMA Channel	Level	26
70	SPI2_RXDMA	SPI2 RX DMA Channel	Level	27
71	SPI2_STAT	SPI2 Status	Level	
72	SPI2_ERR	SPI2 Error	Level	
85	SPI0_TXDMA	SPI0 TX DMA Channel	Level	22
86	SPI0_RXDMA	SPI0 RX DMA Channel	Level	23

Table 15-2: ADSP-2156x SPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
87	SPI0_STAT	SPI0 Status	Level	
88	SPI0_ERR	SPI0 Error	Level	
89	SPI1_TXDMA	SPI1 TX DMA Channel	Level	24
90	SPI1_RXDMA	SPI1 RX DMA Channel	Level	25
91	SPI1_STAT	SPI1 Status	Level	
92	SPI1_ERR	SPI1 Error	Level	
156	SPI2_TXDMA_ERR	SPI2 TX DMA Channel Error	Level	
157	SPI2_RXDMA_ERR	SPI2 RX DMA Channel Error	Level	
158	SPI0_TXDMA_ERR	SPI0 TX DMA Channel Error	Level	
159	SPI0_RXDMA_ERR	SPI0 RX DMA Channel Error	Level	
160	SPI1_TXDMA_ERR	SPI1 TX DMA Channel Error	Level	
161	SPI1_RXDMA_ERR	SPI1 RX DMA Channel Error	Level	

ADSP-2156x SPI Trigger List

Table 15-3: ADSP-2156x SPI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
33	SPI0_TXDMA	SPI0 TX DMA Channel	Edge
34	SPI0_RXDMA	SPI0 RX DMA Channel	Edge
35	SPI1_TXDMA	SPI1 TX DMA Channel	Edge
36	SPI1_RXDMA	SPI1 RX DMA Channel	Edge
37	SPI2_TXDMA	SPI2 TX DMA Channel	Edge
38	SPI2_RXDMA	SPI2 RX DMA Channel	Edge

Table 15-4: ADSP-2156x SPI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
36	SPI0_TXDMA	SPI0 TX DMA Channel	Pulse
37	SPI0_RXDMA	SPI0 RX DMA Channel	Pulse
38	SPI1_TXDMA	SPI1 TX DMA Channel	Pulse
39	SPI1_RXDMA	SPI1 RX DMA Channel	Pulse
40	SPI2_TXDMA	SPI2 TX DMA Channel	Pulse

Table 15-4: ADSP-2156x SPI Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
41	SPI2_RXDMA	SPI2 RX DMA Channel	Pulse

ADSP-2156x SPI DMA Channel List

Table 15-5: ADSP-2156x SPI DMA Channel List

DMA ID	DMA Channel Name	Description
DMA22	SPI0_TXDMA	SPI0 TX DMA Channel
DMA23	SPI0_RXDMA	SPI0 RX DMA Channel
DMA24	SPI1_TXDMA	SPI1 TX DMA Channel
DMA25	SPI1_RXDMA	SPI1 RX DMA Channel
DMA26	SPI2_TXDMA	SPI2 TX DMA Channel
DMA27	SPI2_RXDMA	SPI2 RX DMA Channel

SPI Block Diagram

The *SPI Controller Block Diagram* illustrates the blocks of the SPI module. The module is comprised of three primary parts:

- SPI core contains the receive and transmit FIFOs and their associated shift registers
- Control blocks contain the synchronizer and logic to control the data flow through the data pipelines
- Register block

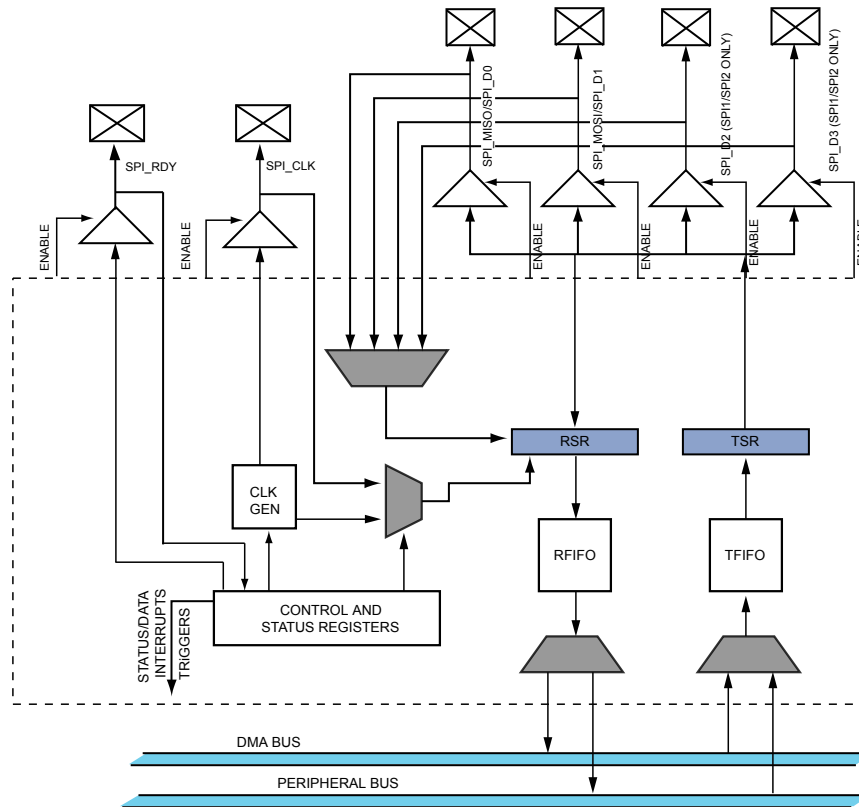


Figure 15-1: SPI Controller Block Diagram, Quad Mode

Transfer Protocol

The SPI module implements two channels that are independent of each other. The SPI module uses the `SPI_RXCTL` and `SPI_TXCTL` dedicated control registers to control these channels. Except in dual and quad modes, SPI can enable and use both channels simultaneously.

The SPI protocol supports four different combinations of serial clock phase and polarity. These combinations are selected through the `SPI_CTL.CPOL` and `SPI_CTL.CPHA` bits.

The *SPI Transfer Protocol* figures demonstrate the two basic transfer formats as defined by the `CPHA` bit. Two waveforms are shown for `SPI_CLK`; one for `SPI_CTL.CPOL=0` and the other for `SPI_CTL.CPOL=1`. The diagrams can be interpreted as master or slave timing diagrams since the `SPI_CLK`, `SPI_MISO`, and `SPI_MOSI` pins are directly connected between the master and the slave. The `SPI_MISO` signal is the output from the slave (slave transmission), and the `SPI_MOSI` signal is the output from the master (master transmission). The master generates the `SPI_CLK` signal. The `SPI_SS` signal is the slave device select input to the slave from the master. The diagrams represent an 8-bit transfer (`SPI_CTL.SIZE=0`) with the MSB first (`SPI_CTL.LSBF=0`). Any combination of the `SPI_CTL.SIZE` and `SPI_CTL.LSBF` bits is permissible. For example, a 16-bit transfer with the LSB first is another possible configuration.

The clock polarity and the clock phase could be identical for the master device and the slave device involved in the communication link. The transfer format from the master can be changed between transfers to adjust for various requirements of a slave device.

The SPI module uses the `SPI_CTL.ASSEL` bit to determine when the SPI hardware or software control the `SPI_SEL[n]` line. When `SPI_CTL.ASSEL=1`, the slave select line must be set to the polarity set in the `SPI_CTL.SELST` field between each serial transfer. The actual behavior of `SPI_SEL[n]` also depends on the parameters programmed into the `SPI_DLY` register. The SPI hardware logic automatically controls this functionality. When `SPI_CTL.ASSEL=0`, `SPI_SEL[n]` can either remain active between successive transfers or be inactive. The software must control this activity through manipulation of the `SPI_SLVSEL` register.

The *SPI Transfer Protocol* pair of figures illustrates the case when `SPI_CTL.ASSEL = 1` and the `SPI_SEL[n]` line is inactive between frames. If `ASSEL = 0`, the `SPI_SEL[n]` line can remain active between frames; however, the first bit is only driven when an active transition of `SPI_CLK` occurs.

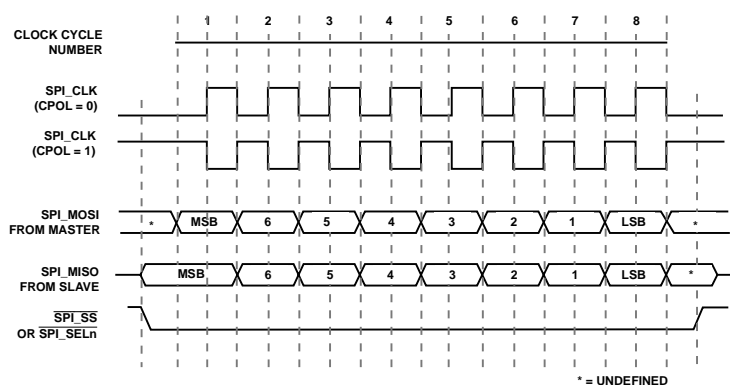


Figure 15-2: SPI Transfer Protocol for CPHA=0

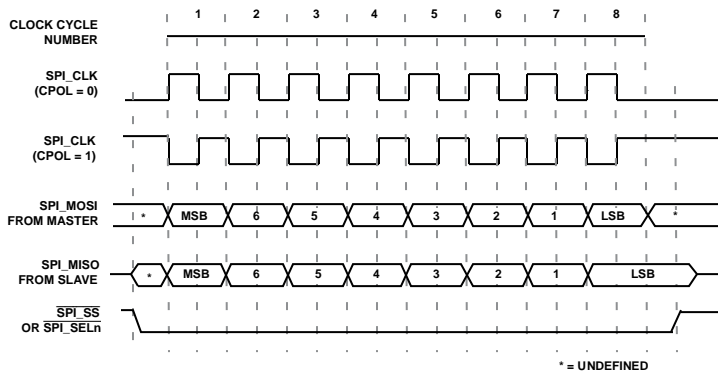


Figure 15-3: SPI Transfer Protocol for CPHA=1

Clock Considerations

The `SPI_CLK` signal is a gated clock that is only active during data transfers, for the time of the transferred word. In normal mode, the number of active edges is equal to the number of bits to be transmitted or received. In dual-I/O mode, it is half of the number of bits to be transmitted or received, and in quad-SPI mode it is one-fourth of the number. The clock rate can be derived using both even and odd dividers from `SCLK`.

For master devices, the SPI uses the `SPI_CLK` register value to determine the clock rate, whereas this value is ignored for slave devices.

When the SPI controller is a master, SPI_CLK is an output signal. Conversely, when the SPI controller is a slave, SPI_CLK is an input signal. Slave devices ignore the SPI clock when the slave select input is driven inactive. The SPI uses the SPI_CLK signal to shift out and shift in the data driven onto the SPI_MISO and SPI_MOSI lines. The data is always shifted out on one edge of the clock (the active edge) and sampled on the opposite edge of the clock (the sampling edge). Clock polarity and clock phase relative to data are programmable through the SPI_CTL register and define the transfer format.

Controlling Delay Between Frames

The *SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)* figure illustrates SPI timing using the SPI_DLY.LEADX and SPI_DLY.LAGX programming. The SPI uses the SPI_DLY.LAGX bits to control the timing between the slave select (SPI_SS) signal assertion and the first SPI_CLK edge. The SPI uses the SPI_DLY.LEADX bits to control the timing between the last SPI_CLK edge and deassertion of the SPI_SS signal. The lead and lag timing can be extended by a 1 SPI_CLK duration to ease timing restrictions on the slave device.

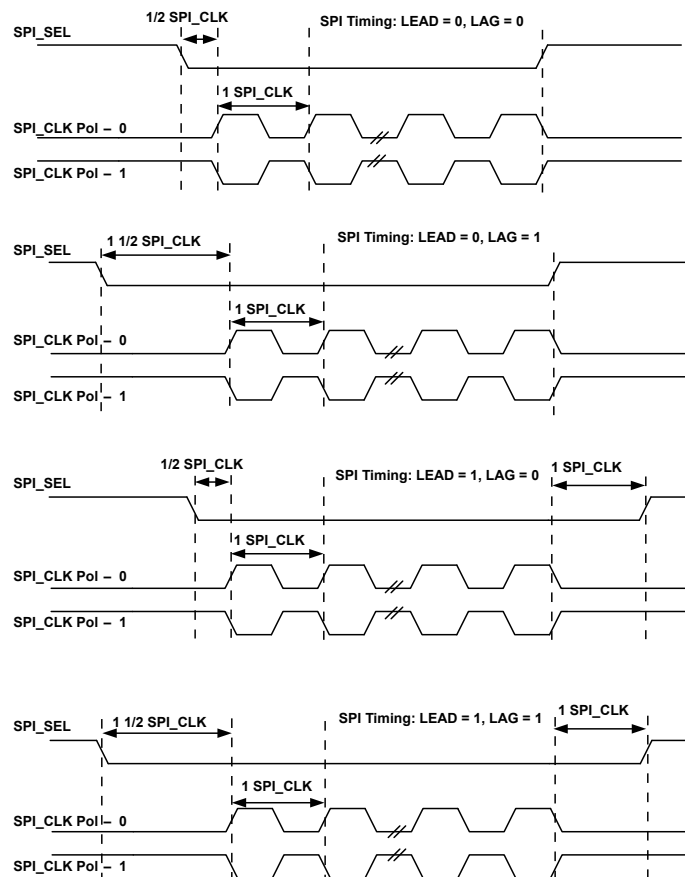


Figure 15-4: SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)

The *SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)* figure illustrates SPI timing with STOP programming. The SPI module uses this timing to insert multiples of SPI_CLK period

delays between transfers. The `SPI_SS` line is deasserted for the duration specified in the `SPI_DLY.STOP` bit field, assuming the `SPI_CTL.SELST` bit is configured for deassertion between transfers.

If the `SPI_DLY.STOP` bit = 0, the master operates in a *continuous mode*. This mode causes an immediate start of the second word after the last bit is transferred from the first word. During this mode of operation, the slave select line is continuously asserted.

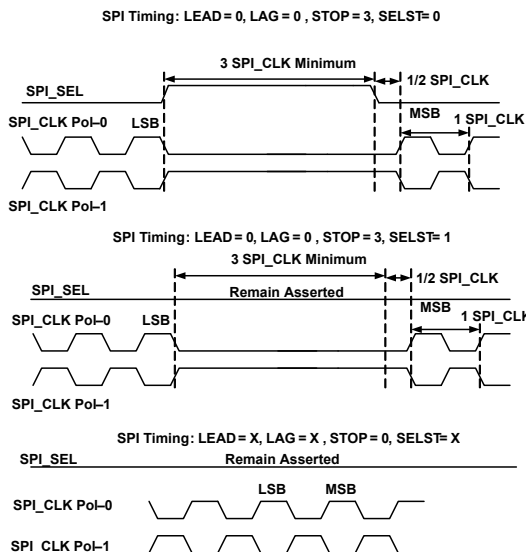


Figure 15-5: SPI Timing with `SPI_DLY.STOP` Programming (Independent of `SPI_CTL.CPHA` Setting)

When the `SPI_DLY.STOP` bit = 0 and initial conditions for a transfer are not met, the interface pauses before the next transfer. During this pause, the SPI uses the `SPI_CTL.SELST` bit to determine the state of the slave select pin. The SPI uses the `SPI_DLY.LEADX` and `SPI_DLY.LAGX` bits to determine the timing between `SPI_CLK` edges and the slave select line.

Flow Control

In master mode, the slave device must drive the `SPI_RDY` pin. The pin acts as an input signal. The slave can deassert the `SPI_RDY` pin to stop the master from initiating any new transfer. If `SPI_RDY` is deasserted in the middle of a transfer, the current transfer continues, and the next transfer will not start unless the slave asserts the `SPI_RDY` signal. Whenever the slave deasserts `SPI_RDY` and stalls the master, the SPI controller goes into a waiting state, and the `SPI_STAT.FCS` bit is set. When the slave asserts `SPI_RDY`, the SPI controller resumes operation, and the `SPI_STAT.FCS` bit is cleared.

In slave mode, the `SPI_RDY` pin acts as an output signal. Flow control can be configured on either the TX channel or the RX channel. The SPI uses the `SPI_CTL.FCCH` bit to control this configuration. If flow control is configured on the TX channel, as the `SPI_TFIFO` status nears the empty condition, the `SPI_RDY` pin is deasserted. If flow control is configured on the RX channel, as the `SPI_RFIFO` status nears the full condition, the `SPI_RDY` pin is deasserted. The SPI uses the `SPI_CTL.FCWM` bits to control the FIFO status at which `SPI_RDY` deassertion takes place. Flow control in slave mode is purely based on the FIFO status and does not depend on the word counters.

The *SPI Flow Control Timing in Master Mode* figure illustrates this timing.

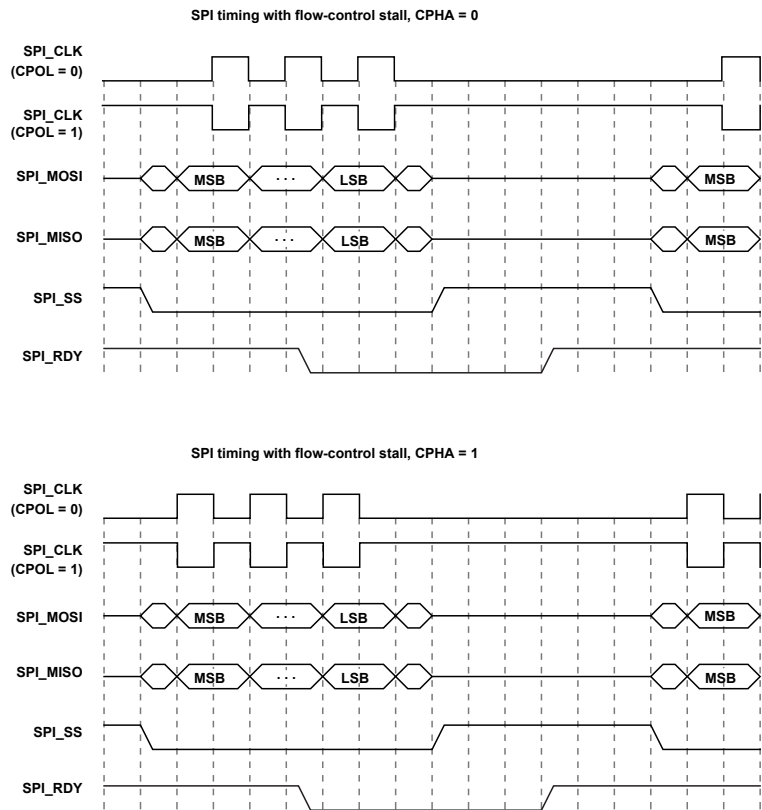


Figure 15-6: SPI Flow Control Timing in Master Mode.

Slave Select Operation

If the SPI is in slave mode, `SPI_SS` acts as the slave select input. When SPI is enabled as a master, `SPI_SS` can serve as an error detection input for the SPI in a multi-master environment. The `SPI_CTL.PSSE` bit enables this feature. When `SPI_CTL.PSSE=1`, the `SPI_SS` input is the master mode error input. Otherwise, `SPI_SS` is ignored.

The `SPI_SS` signal is an active-low signal. The master asserts the signal during the transfer. The signal can be deasserted or remain asserted between transfers. When `SPI_SS` is deasserted, `SPI_CLK` and inputs are ignored, and outputs are three-stated.

The slave select bits (`SPI_SLVSEL.SSEL1 – SPI_SLVSEL.SSEL7`) are used in a multiple-slave SPI environment. For example, if there are eight SPI devices in the system including a processor master, the master processor can support the SPI mode transactions across the other seven devices. This configuration requires only one master processor in this multi-slave environment.

For example, assume that the SPI of the processor is the master. The `SPI_SLVSEL.SSEL1 – SPI_SLVSEL.SSEL7` bits on the processor can be connected to the slave select pin of each slave device. In this configuration, the slave select bits can be used in three ways. In cases 1 and 2, the processor is the master and the seven microcontrollers or peripherals with SPI interfaces are slaves. The processor can do one of the following:

1. Transmit to all seven SPI devices at the same time in a broadcast mode. Here, all slave select bits are set.
2. Receive and transmit from one SPI device by enabling only one slave SPI device at a time.
3. If all the slaves are also processors, then the requester can receive data from only one processor at a time. (The functionality is enabled by clearing the `SPI_CTL.EMISO` bit in the six other slave processors.) The requester can transmit broadcast data to all seven at the same time. This MISO enabling feature is available in some other microcontrollers. Therefore, it is possible to use the MISO enabling feature with any other SPI device that includes this functionality.

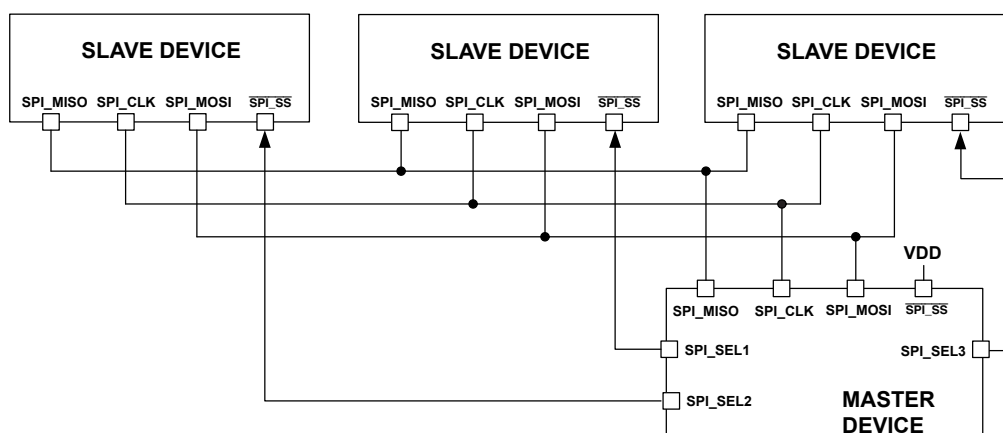


Figure 15-7: Single-Master, Multiple-Slave Configuration

Beginning and Ending a Non-DMA SPI Transfer

The start and finish of a non-DMA SPI transfer depend on the following settings.

1. Whether the device is configured as a master or a slave.
2. The state of the `SPI_CTL.ASSEL` bit, which selects between hardware and software control over `SPI_SLVSEL`.

When `SPI_CTL.CPHA=0`, the enabled slave select outputs are driven active. However, the `SPI_CLK` signal remains inactive for the first half of the first cycle of `SPI_CLK`. For a slave with `SPI_CTL.CPHA=0`, the transfer starts as soon as the `SPI_SS` input goes low.

When `SPI_CTL.CPHA=1`, a transfer starts with the first active edge of `SPI_CLK` for both slave and master devices. For a master device, a transfer is complete after it sends the last data and simultaneously receives the last data bit. A transfer for a slave device ends after the last sampling edge of `SPI_CLK`. If `SPI_CTL.ASSEL=1`, the hardware maintains responsibility for toggling `SPI_SS` between frames. If `SPI_CTL.ASSEL=0`, software controls the `SPI_SS` line and can keep it active between frames.

The `SPI_STAT.RFE` bit defines when the receive buffer can be read, indicating that `SPI_RFIFO` is not empty. The `SPI_STAT.TFF` bit defines when the transmit buffer can be written, indicating that the `SPI_TFIFO` is not full. The end of a single word transfer occurs when the `SPI_STAT.RFE` bit is cleared. The status indicates that a new word has been received and written into the receive FIFO. The `SPI_STAT.RFE` bit remains cleared as long as the receive FIFO has valid data.

To maintain software compatibility with other SPI devices, the `SPI_STAT.SPIF` bit is also available for polling. This bit can have a slightly different behavior from other commercially available devices.

In master mode with the `SPI_CTL.ASSEL` bit cleared, software manually asserts the required slave select signal before starting the transaction. After all data transfers, software typically releases the slave select line.

When the receive or transmit word counters are enabled in the `SPI_TXCTL` or `SPI_RXCTL` registers, the SPI generates a finish interrupt at the end of the transfer. It signals the end of all transfers related to that transaction.

Transmit Operation in Non-DMA Mode

The transmit operation in non-DMA mode is enabled through the `SPI_TXCTL.TEN` bit. It can be enabled independently from the receive operation, and the transmit channel can become the initiating channel based on the `SPI_TXCTL.TTI` bit setting.

Transmit underrun is not possible in this mode, as no new transfer initiates unless the transmit FIFO is empty (in the case that `SPI_TXCTL.TTI = 1`). A receive overflow is detected when data from a new frame transfer replaces older data in a full receive FIFO. This event can occur if `SPI_TXCTL.TTI = 1` and the receive channel is enabled in a non-initiating capacity.

An SPI transmit interrupt request is signaled once the transmit channel has been enabled and the transmit FIFO is not full. The SPI uses the `SPI_TXCTL.TDR` bit setting to control the frequency of the interrupt request.

Receive Operation in Non-DMA Mode

The receive operation in non-DMA mode is enabled through the `SPI_RXCTL.REN` bit. It can be enabled independently from the transmit operation, and the receive channel can become the initiating channel based on the `SPI_RXCTL.RTI` bit setting.

Receive overflow is not possible in this mode, as no new transfer initiates when the receive FIFO is full (in the case of `SPI_RXCTL.RTI = 1`). A transmit underrun can occur (`SPI_TXCTL.TDU` bit) when no valid data is in the `SPI_TFIFO` register when a transfer is initiated. This event can occur if `SPI_RXCTL.RTI = 1` and the transmit channel is enabled in a non-initiating capacity.

An SPI receive interrupt request is signaled once the receive channel has been enabled and there is data waiting to be read. The SPI uses the `SPI_RXCTL.RDR` bit setting to control the frequency of the interrupt request.

Dual I/O Mode

In Dual I/O mode, the `SPI_MISO` and `SPI_MOSI` pins are configured to operate in the same direction which doubles bandwidth. The SPI uses the `SPI_CTL.SOSI` bit to determine the order of bits on the pins. When set, the processor sends the first bit on the `SPI_MOSI` pin and the second bit on the `SPI_MISO` pin. If the `SPI_CTL.SOSI` bit is cleared, the order is reversed. Since dual I/O mode uses both pins to transmit or receive data, only one channel can be enabled, either transmit or receive. Flow control through the `SPI_RDY` pin is supported. Interrupt request generation is unaffected by dual I/O mode. However, the interrupt service interval is reduced, since the individual transfer latency is halved.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

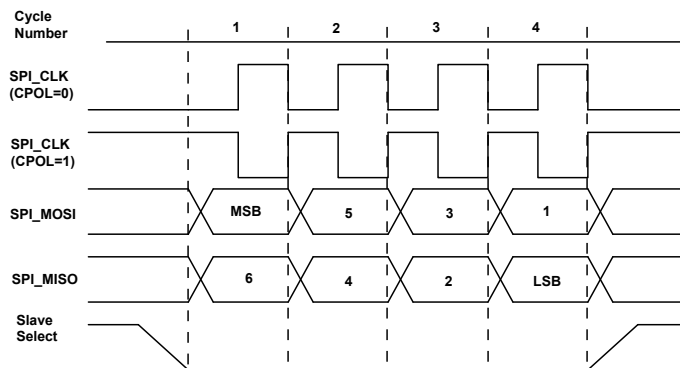


Figure 15-8: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

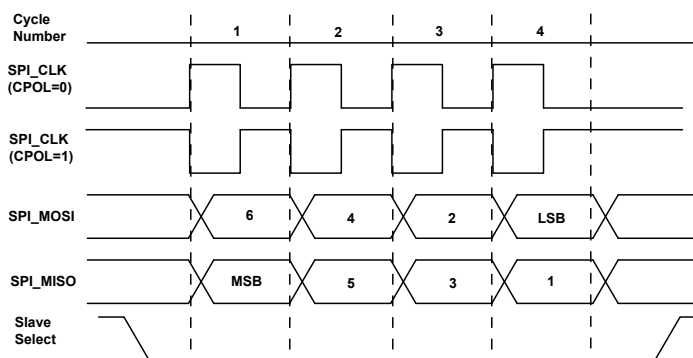


Figure 15-9: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

Quad I/O Mode (SPI1/SPI2 only)

In quad SPI mode, the SPI_MISO and SPI_MOSI pins, in tandem with the SPI_D2 and SPI_D3 pins, are configured to operate in the same direction. The SPI uses the SPI_CTL.SOSI bit to determine the order of bits on the pins. When set, the processor sends:

- The first bit on the SPI_MOSI pin
- The second bit on the SPI_MISO pin
- The third bit on the SPI_D2 pin
- The fourth bit on the SPI_D3 pin

If the SPI_CTL.SOSI bit is cleared, the order is reversed. Since quad SPI mode uses all four pins to transmit or receive data, only one channel can be enabled as either transmit or receive. Flow control through the SPI_RDY pin is supported. Interrupt generation is unaffected by quad SPI mode.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

While using dual or quad I/O mode for communicating with flash devices, program the SPI_CTL.CPHA and the SPI_CTL.CPOL bits =1. This programming avoids bus contention during read operations, because the flash device starts driving out the bits immediately after dummy cycles in read header.

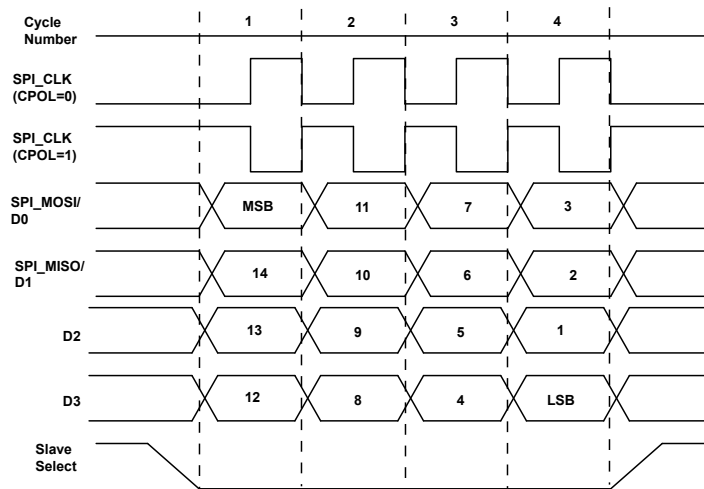


Figure 15-10: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

NOTE: The SPI does support quad SPI 8-bit transfer in slave continuous mode of operation with an SCLK:SPI_CLK ratio of less than 1:2. A minimum of 2 SCLK cycles is required between transfers in 8-bit quad SPI slave mode with an SCLK:SPI_CLK ratio of less than 1:2.

Fast Mode

Fast mode is similar to the normal mode of operation when transmitting. When receiving, data is sampled at the next transmit edge allowing a full cycle of timing in the receive direction. This mode is valid in master mode operation only. When the SPI operates in fast mode, the slave drives the data for one full cycle.

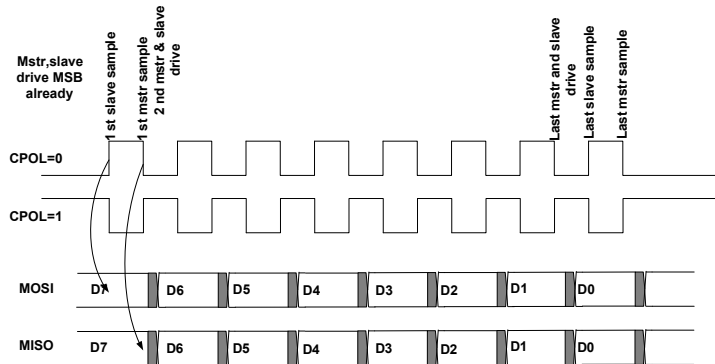


Figure 15-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0

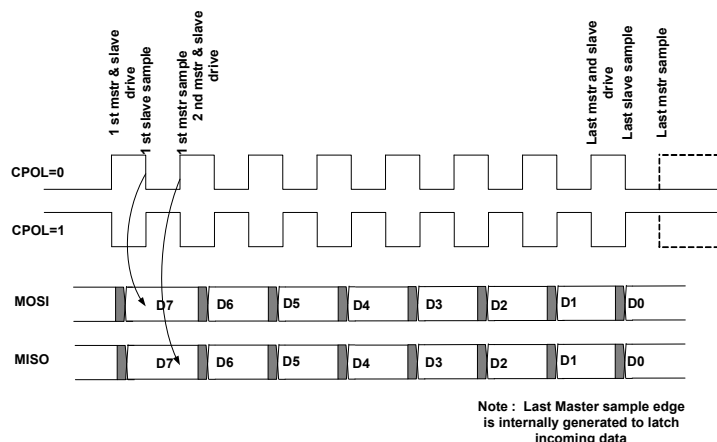


Figure 15-12: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

Memory-Mapped Mode (SPI2 only)

The SPI supports direct memory-mapped read accesses from a SPI memory device, enabled by setting the SPI_CTL.MMSE bit. This mode allows for direct execution of instructions from a SPI memory device without the need for a low-level software driver, as hardware handles all overhead tasks (for example, transmission of the read header, pin turnaround timing, and receive data sizing). The SPI features configurable options in the memory-mapped read header register (SPI_MMRDH) to provide compatibility with a wide range of SPI memory devices.

In non-memory-mapped mode, the software is responsible for providing the command and required dummy words for the read response, whereas this is all handled by hardware when the SPI is in memory-mapped mode. The memory of the SPI device is accessible directly through reads of the processor address space. The read accesses can be code or data accesses in core mode or when using memory DMA (MDMA). These accesses allow code to execute directly from SPI memory devices (true eXecute-In-Place operations), and the contents can be cached to improve performance. It is not necessary to access the SPI data buffer registers nor poll status bits; however, the hardware does not support peripheral DMA accesses nor write operations to the SPI memory space.

The *Types of Operations* table is a comparison of the permitted operations in the non-memory-mapped and memory-mapped modes supported by the SPI controller.

Table 15-6: Types of Operations

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
Core data write	Yes	No
Core data read	Yes	Yes
Code fetch: Execute-In-Place (XIP)	No	Yes
Read/Write accesses using SPI Peripheral DMA	Yes	No
Read/Write accesses by other peripheral DMA channels	No	No
MDMA read	No	Yes

Table 15-6: Types of Operations (Continued)

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
MDMA write	No	No

Memory-Mapped Description of Operation

Memory-mapped mode is enabled by setting the `SPI_CTL.MMSE` bit. When enabled, the SPI (if ready) accepts the read requests through a dedicated on-chip slave interface. The memory subsystem master drives this dedicated interface through the SCB fabric.

In a typical scenario, the memory subsystem master issues read requests to the fabric, and the fabric routes these requests to the slave port of the SPI peripheral. The master describes the read access using a number of parameters such as starting address, transfer size, and burst type. The SPI responds to this read access request when it is ready for a new transfer. It loads the opcode, a specified number of address bytes, and an optional mode byte into the transmit FIFO. The SPI memory state machine begins when both the transmit and receive channels of the SPI are enabled with:

- the transmit transfer initiation bit is set (`SPI_TXCTL.TTI=1`)

The SPI memory read sequence starts with the assertion of `SPI_SEL1`. If the SPI memory state machine is in the reset state, it looks for a command. The SPI hardware then sends the specific 8-bit read command (which can be optionally skipped), followed by the SPI memory read address. After this, a dummy period is inserted, in which a mode byte is optionally sent and the pins are held or three-stated during the dummy clocking period.

NOTE: This read header is transmitted over the SPI standard protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, `SPI_SEL1`) or over the extended SPI protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, `SPI_D2`, `SPI_D3`, `SPI_SEL1`), based on the `SPI_MMRDH.COMDPINS`, `SPI_MMRDH.ADRPINS`, and `SPI_CTL.MIOM` bit settings. SPI memory devices usually support communication in MSB-first mode. In dual mode, the SPI typically uses `SPI_MISO` as IO1 and `SPI_MOSI` as IO0. In quad mode, the SPI typically uses `SPI_D3` pin as IO3, `SPI_D2` as IO2, `SPI_MISO` as IO1, and `SPI_MOSI` as IO0.

When all I/O data pins are three-stated, the SPI continues clocking the SPI memory device, which drives out the data bits at the addressed location, until all bytes are received. The SPI hardware reads the data as configured by the `SPI_CTL.MIOM` bit setting. Upon reception of the last byte, the SPI typically deasserts `SPI_SEL1` to prepare for the next requested read header.

Application code must ensure that the opcode sent is consistent with multiple I/O programming and that the parameters specified in the memory-mapped read header register are consistent with flash read access timing.

The *SPI Memory-Mapped Register Operations Flow* diagram shows how the fields of the `SPI_MMRDH` register determine the read header while initiating transfers in memory-mapped mode.

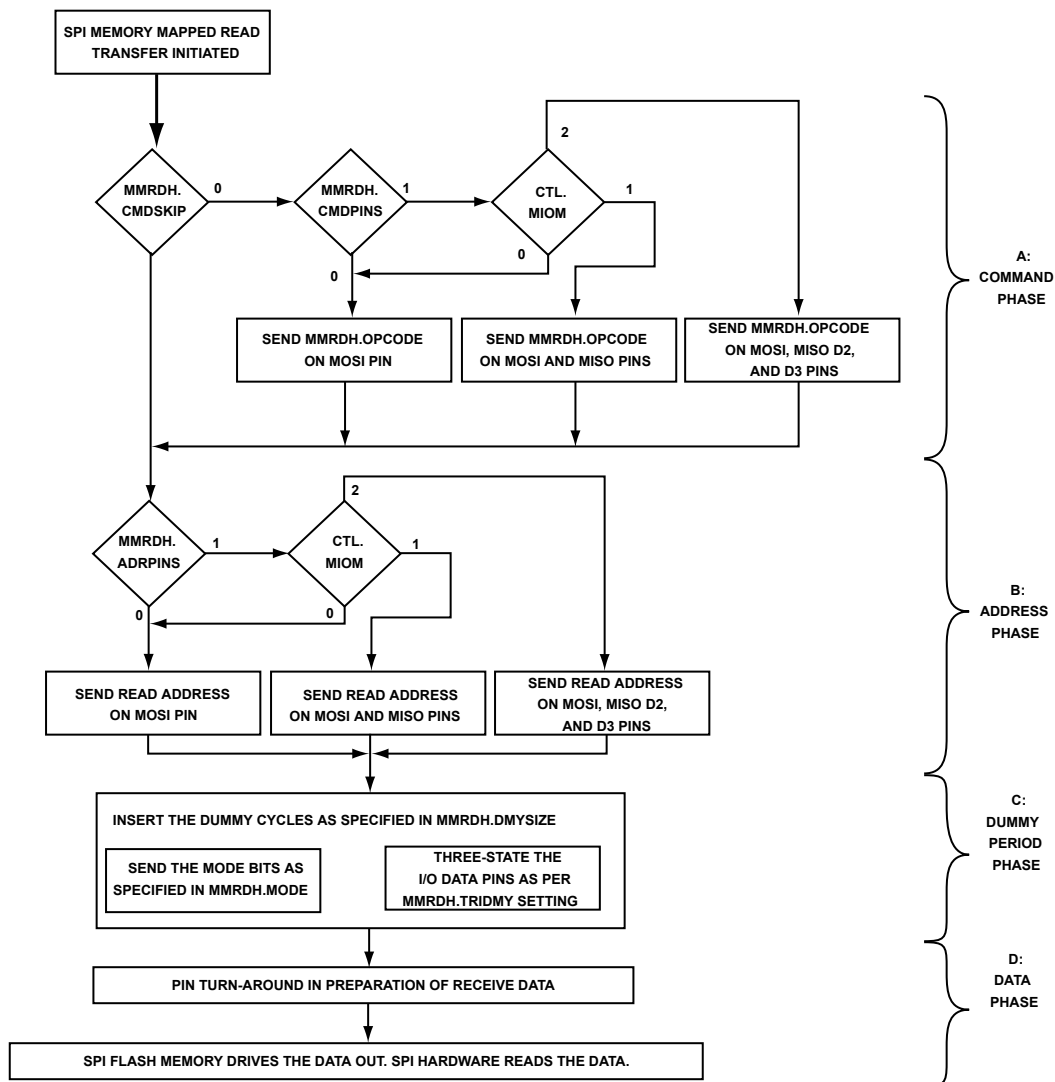


Figure 15-13: SPI Memory-Mapped Register Operations Flow

Memory-Mapped Architectural Concepts

In memory-mapped mode, the SPI accepts read requests through a dedicated on-chip slave interface. The SPI (if ready) accepts these requests and begins the process of assembling the read header based on access attributes described in both the `SPI_MMRDH` register and the internal bus request. After the read header transmission is complete, a pin turnaround period is timed and the receiver is enabled. The SPI continues clocking the SPI memory device until all bytes are received.

The SPI memory-mapped hardware accommodates various memory devices with different read timing. The capabilities include extra mode bits, flexible dummy period timing, and three-state control, as configured in the `SPI_MMRDH` register.

The *Memory-Mapped Protocol* figure shows the protocol for the SPI controller in memory-mapped mode.

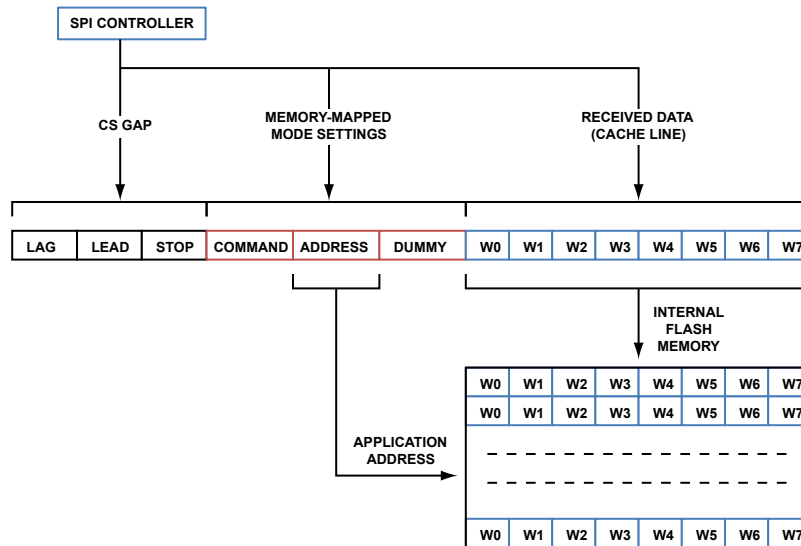


Figure 15-14: Memory-Mapped Protocol

As shown in the figure, the COMMAND field (`SPI_MMRDH.OPCODE`) is transmitted upon assertion of the `SPI_SEL[n]` signal. The SPI memory interprets this 8-bit value as a read command. Any 8-bit read opcode whose timing is compliant with the processor SPI and features provided by memory-mapped hardware is allowed, the most common being:

- Standard Read (0x03)
- Fast Read (0x0B)
- Fast Read Dual Output (0x3B)
- Fast Read Dual I/O (0x6B)
- Fast Read Quad Output (0xBB)
- Fast Read Quad I/O (0xEB)

NOTE: The SPI hardware does not validate the content of the `SPI_MMRDH.OPCODE` field prior to transmitting.

DMYSIZE (Number of Dummy Bytes)

When operating at a high clock frequency in multi-IO modes, most flash devices require some dummy clocks after the address bits. These dummy clock cycles allow the internal circuits of the device extra time for setting up the initial address. These bits specify the number of bytes separating address transmission and read data return.

The number of dummy cycles required varies per manufacturer, the read command used, and the SPI access time. The SPI hardware allows dummy cycles to be programmed in bytes in the `SPI_MMRDH.DMYSIZE` field, the value of which is a function of the number of pins used to transmit the address (`SPI_MMRDH.ADRPINS`), as shown in the *Pins Used to Transmit the Address (ADRPINS)* table.

Table 15-7: Pins Used to Transmit the Address (ADRPINS)

SPI_MMRDH. DMYSIZE	Dummy clock cycles		
	(SPI_MMRDH.ADRPINS=0, SPI_CTL.MIOM=x) Dummy bytes elapse over 1-pin	(SPI_MMRDH.ADRPINS=1, SPI_CTL.MIOM=1) Dummy bytes elapse over 2- pins	(SPI_MMRDH.ADRPINS=1, MIOM=2) Dummy bytes elapse over 4- pins
000	0	0	0
001	8	4	2
010	16	8	4
011	24	12	6
100	32	16	8
101	40	20	10
110	48	24	12
111	56	28	14

This dummy clocking period allows the mode bits to be sent, the pins to be three-stated, and the pins to be turned around in preparation for the receive data.

Memory-Mapped Read Accesses

The SPI hardware supports the most commonly used read operations.

- Two standard SPI reads (read and read fast), which use the unidirectional SPI_MOSI and SPI_MISO pins in addition to SPI_SEL[n] and SPI_CLK
- Four extended SPI multiple I/O reads: dual output, quad output, dual I/O, and quad I/O reads

The *SPI Read Operations* table and *SPI Flash Fast Read Sequence* figures summarize the types of read operations. Program each read operation in a way that is compatible with the description given in the SPI flash data sheet.

Table 15-8: SPI Read Operations

Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Read	0x03	1	1	Zero	No	No	1
Fast Read	0x0B	1	1	Non-Zero	Yes	No	1
Dual Output Read	0x3B	1	1	Non-Zero	Yes	Yes(IO0-1)	2
Quad Output Read	0x6B	1	1	Non-Zero	Yes	Yes(IO0-3)	4
Dual I/O Read	0xBB	1, 2	2	Non-Zero	Yes	Yes (IO0-1)	2
Quad I/O Read	0xEB	1, 4	4	Non-Zero	Yes	Yes (IO0-3)	4

Some memory devices also support word quad I/O read (0xE7) and octal quad I/O read (0xE3) operations. These operations require fewer dummy cycles than normal quad I/O read operations.

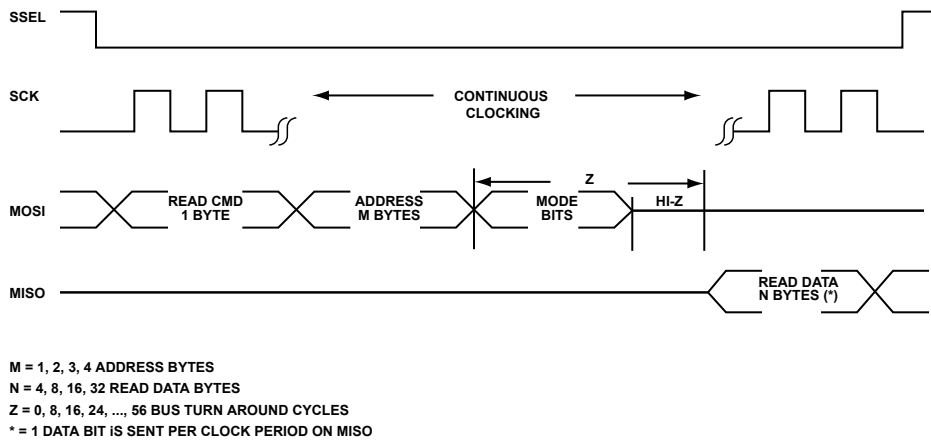


Figure 15-15: SPI Flash Fast Read Sequence

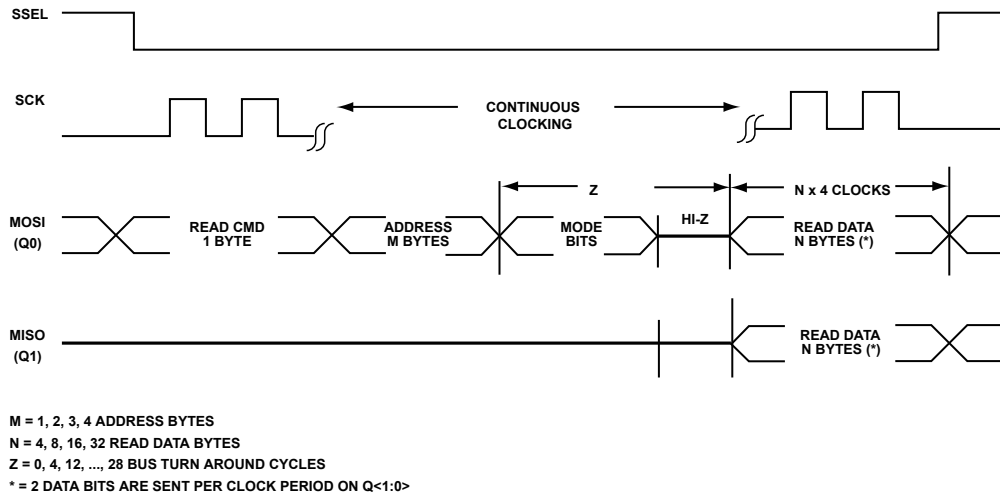


Figure 15-16: SPI Flash Fast Read (Dual Output) Sequence

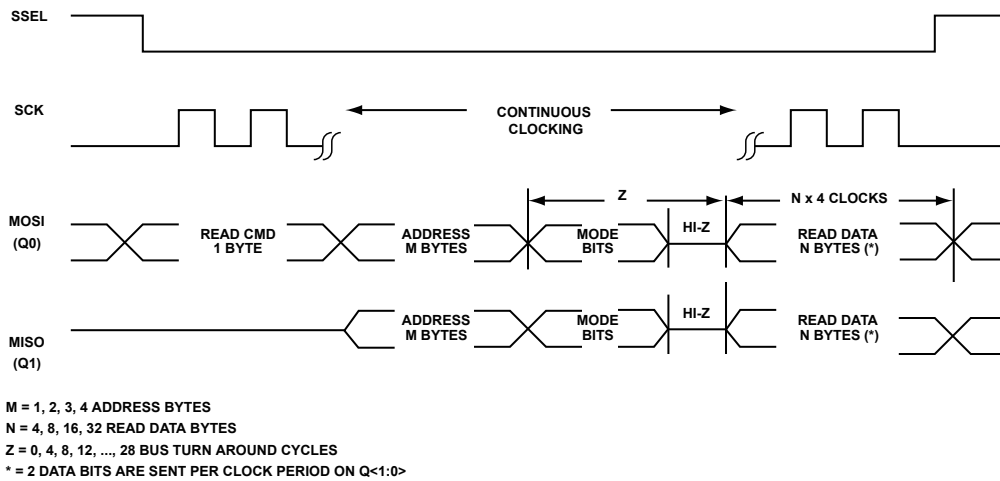


Figure 15-17: SPI Flash Fast Read (Dual I/O) Sequence

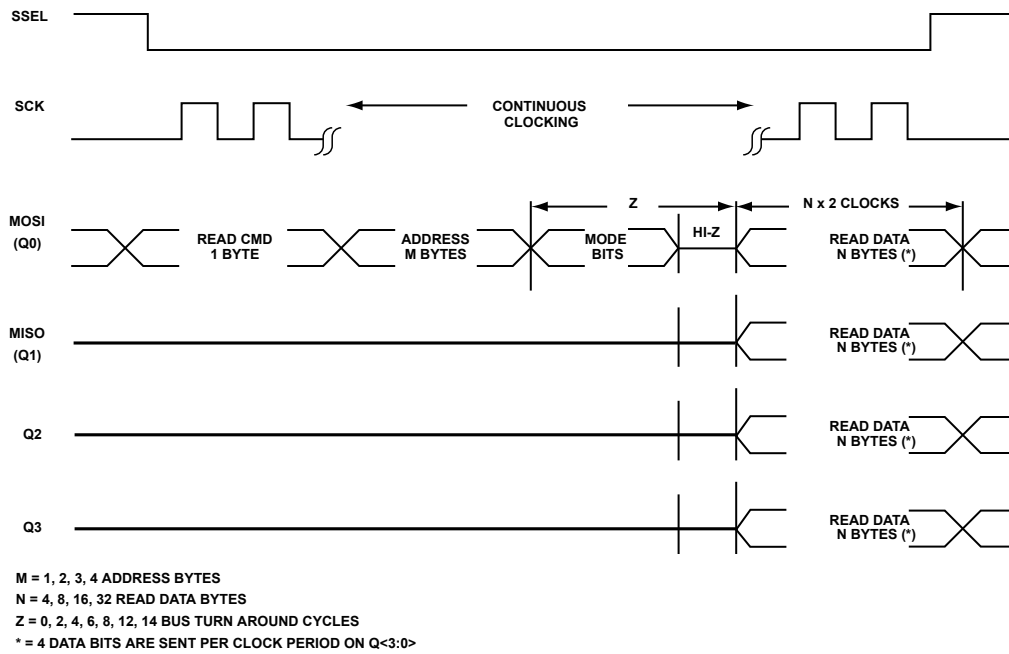


Figure 15-18: SPI Flash Quad Output Read Sequence

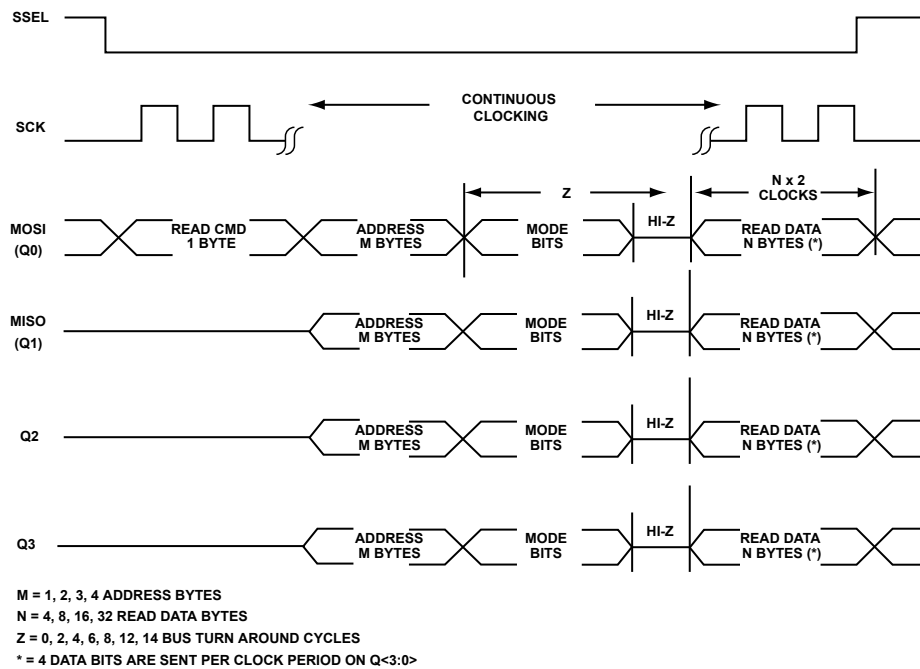


Figure 15-19: SPI Flash Quad I/O Read Sequence

SPI memory-mapped reads can be made cacheable in the core's internal memory by properly configuring the region as cacheable memory without bypass (see the related core's cache configuration documentation for details). In the figures, the number of read data bytes (N) is based on the following:

- For an instruction fetch by core (when in XIP mode); the number of instruction bytes to be fetched depends on the cache line size of the cache.

- For a data fetch by the core (data read), the number of data bytes to be fetched depends on the cache line size of the cache.

Although the minimum size of a memory-mapped data read transfer is 4 bytes, applications can fetch a single byte or a 2-byte data. (For example, it can fetch an unsigned char or short access in C code). In this case, only the required bytes are provided to the core and the other bytes are cached.

The on-chip memory subsystem master provides a starting address for the burst and the SPI hardware issues this address as part of the read header. The address provided is N-byte aligned. For example, to read the 30th byte from SPI memory, then the typical address to provide is:

- 28 (0x0000_001C) for a 32-bit cache line
- 24 (0x0000_0018) for a 64-bit cache line
- 16 (0x0000_0010) for a 128-bit cache line
- 0 (0x0000_0000) for a 256-bit cache line

The read data is returned to the memory subsystem in the order provided by the SPI memory. There can be considerable delay for the expected data provided to the master.

To minimize this delay, the wrap feature can be used where the memory subsystem provides the address of the critical word.

- For MDMA reads, the number of read data bytes (N) is always equal to 4 bytes. The MDMA read does not depend on the cache setting. For MDMA reads, limit the `DMA_CFG.MSIZE` field to 1, 2 or 4 bytes. The address provided by the memory subsystem master to the SPI hardware is always 4 byte-aligned.

Memory-Mapped High-Performance Features

In addition to automating the SPI memory read accesses, the memory-mapped hardware also provides some features to improve SPI memory fetches and increase the system performance. The following sections describe these features.

Merged Read Accesses

It is common for the memory subsystem to fetch two or more cache lines from consecutive addresses (the address sequencing is linear without any jumps). To take advantage of this situation, the SPI memory-mapped hardware provides a feature called merging. Enable merging by setting the `SPI_MMRDH.MERGE` bit.

When enabled, the hardware compares the address of an incoming read request to the address of a request the SPI memory is actively servicing. It can decide to merge two accesses when the address for the second access is incremental. For example, if the first address of a 32-byte cache line fetch is 0x0000_0000 and the second fetch is to address 0x0000_0020, then these two accesses can be merged. Merging increases efficiency and overall fetch bandwidth by eliminating the read header for those accesses which only require continuation of the SPI clock.

Wrap Around Accesses

Many SPI flash memory devices support wrapping which is used to enhance critical word fetching of cache lines. In this mode, the SPI device automatically wraps the read address to the base of a cache line once the end of the cache line is reached.

Wrap around accesses are enabled by setting the `SPI_MMRDH.WRAP` bit.

Some flash devices require programs to send a *Set Wrap* command to place the device in wrap mode. Other flash devices provide a configuration register which must be programmed to set the flash in wrap mode. Since the SPI memory-mapped hardware does not support any write operations to flash, perform this step in non-memory-mapped mode (`SPI_CTL.MMSE=0`) by accessing the SPI registers.

Data access is limited to 8-byte, 16-byte, or 32-byte sections of flash page in wrap mode. The Arm core uses the Wrap 4 access (64-bit data) for L1 cache. The Arm core uses Wrap 4 and Wrap 8 accesses (64-bit data) for L2 cache. The cores use the Wrap 8 accesses for unaligned accesses. During the read request to the SPI memory-mapped hardware, the memory subsystem master of the processor provides the address of a critical word instead of the line base. The read-data starts at the address specified in the instruction. Once it reaches the end boundary of the 8, 16, or 32-byte section, the output automatically wraps around to the beginning boundary to the line base address. The data fetch continues. It is not necessary to deassert the SPI `SPI_SEL[n]` signal or resend the read header to wrap to the cache line base when servicing misaligned cache fill requests.

The *Byte Sequence in Wrap Modes* table shows byte sequences in various wrap modes.

Table 15-9: Byte Sequence in Wrap Modes

Starting Address	8-Byte Wrap (cache_line = 8 byte)
0	0-1-2- . . . -6-7
1	1-2- 3-. . . -7-0
7	7-0-1- . . . -5-6
15	15-8-9- . . . -13-14
31	31-24-25-. . . -29-30

The burst with wrap feature allows applications to fetch a critical address quickly. Applications then fill the cache afterwards within a fixed length (8/16/32-byte) of data without issuing multiple read commands. Certain applications can benefit from this feature to improve cache fill efficiency and overall performance of system code execution.

NOTE: Do not use the merge and wrap feature together. Using wrap bursts can unintentionally disable merging (merging cannot occur for unaligned wrapping bursts). A wrap burst can start fetching data words in the middle of the cache line and cannot be merged with the next access.

Execute-In-Place (XIP, SPI2 only)

Execute-In-Place, most commonly known as XIP, allows software code to execute directly from an SPI flash device rather than downloading the code and executing it out of RAM. XIP, also known as Command Skip mode, is a general term and can be applied to fetching data as well.

There is a difference between XIP mode and standard mode. In XIP mode, after the SPI memory device is selected (`CS# =LOW`), the memory device does not decode the first input byte as command code. Instead, it expects the read header to directly start with address bytes. In standard mode, the memory decodes the first input byte it receives as a command code.

The XIP mode dramatically reduces random access time for applications that require fast code execution without shadowing the memory content on a RAM. The SPI memory-mapped hardware provides a control bit, `SPI_MMRDH.CMDSKIP` to skip the command from read header.

Some SPI memory devices require configuration of their control register to enable the XIP mode of operation, using the non-memory-mapped mode of the processor SPI. Typically, during the dummy cycle period, the mode bits are used to confirm the XIP operation and the `SPI_MMRDH.MODE` field must be set appropriately. A dummy memory-mapped access may be needed before setting the `SPI_MMRDH.CMDSKIP` bit in order to set the SPI memory device in Command Skip mode.

For more details about how to configure SPI memories into XIP mode, refer to the device data sheet.

NOTE: When configuring the flash to XIP mode from the SHARC+ core, ensure that the routine that configures flash to XIP is not routed through the L2CC. This is accomplished by first configuring the flash to XIP mode, then enabling the L2CC from the core.

Memory-Mapped Mode Error Status Bits

The SPI memory-mapped hardware provides bits in the `SPI_STAT` register to report errors. It provides these bits for notification only and their state has no effect on SPI operations. The status register bits are sticky. A W1C (write-1-to-clear) operation clears the bits.

- Memory-Mapped Write Error (`SPI_STAT.MMWE`). This bit is set (=1) if an attempt is made to write to address space that is reserved for memory-mapped SPI memory. The SPI memory-mapped hardware does not support automated write access to SPI memory space.
- Memory-Mapped Read Error (`SPI_STAT.MMRE`). This bit is set (=1) if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (`SPI_CTL.MMSE = 0`).
- Memory-Mapped Access Error (`SPI_STAT.MMAE`). This bit is set (=1) if an attempt is made to access either the TX or RX FIFO while memory-mapped access of SPI memory is enabled. In this case, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric reports an error. Legacy methods include any direct access made to the TX and RX FIFOs, whether by DMA or processor MMR.
- Memory-Mapped Write Error Mask (`SPI_CTL.MMWEM`) bit specifies whether an error response is returned to the fabric on write attempts to address space that is reserved for memory-mapped SPI memory reads. Regardless of whether a write error response is masked using this bit, the memory-mapped write error (`SPI_STAT.MMWE`) sticky notification bit is still set.

NOTE: Unlike other bits in the `SPI_STAT` register, these memory-mapped mode error bits do not have associated bits in the SPI interrupt mask (`SPI_IMSK`) and SPI interrupt condition (`SPI_ILAT`) registers.

The memory-mapped top register (`SPI_MMTOP`) is used to specify the upper limit of the SPI memory address. The memory-mapped accesses to SPI memory addresses equal to or above this range are considered illegal. The accesses are blocked and a bus error response is generated.

This register is useful to block the invalid SPI memory address accesses. Some SPI memory vendors do not clearly specify (guarantee) that overrange address bits are ignored (address spaces can be wrapped).

Memory-Mapped Programming Guidelines

Setting the `SPI_CTL.MMSE` bit enables SPI memory-mapped mode. When enabled, the SPI interface is forced to be consistent with SPI memory requirements regardless the settings of certain control bits. The following tables specify typical settings for configuring the SPI in memory-mapped mode:

Table 15-10: SPI Control (SPI_CTL) Register

Bits	Typical values to set	Description	Comments
<code>SPI_CTL.MSTR</code>	1	Master mode enable	
<code>SPI_CTL.PSSE</code>	0	Protected slave select enable	
<code>SPI_CTL.ODM</code>	0	Open-drain mode enable	
<code>SPI_CTL.CPHASPI_CTL.CPOL</code>	0–0 or 1–1	SPI mode of communication	Flash dependent, usually SPI flash supports mode-0 (CPHA=CPOL=0) and mode-3 (CPHA=CPOL=1)
<code>SPI_CTL.ASSEL</code>	1	Hardware slave select pin control	
<code>SPI_CTL.SELST</code>	1	Slave select asserted between transfers	
<code>SPI_CTL.EMISO</code>	1	MISO pin enable	
<code>SPI_CTL.SIZE</code>	2	32-bit transfer size	
<code>SPI_CTL.LSBF</code>	0	MSB bit first mode	Flash dependent, usually SPI flash communicates in MSB bit first mode
<code>SPI_CTL.FCEN</code> <code>SPI_CTL.FCCH</code> <code>SPI_CTL.FCPL</code> <code>SPI_CTL.FCWM</code>	0	Hardware flow control related bits	
<code>SPI_CTL.FMODE</code>	1	Fast mode enable	Typically set to 1 for full cycle timing, 0 only works at low speed
<code>SPI_CTL.SOSI</code>	0	Treat <code>SPI_MOSI</code> pin as IO0 pin.	

Table 15-11: SPI Receive Control Register

Bits	Typical values to set	Description
<code>SPI_RXCTL.REN</code>	1	Receive channel enable
<code>SPI_RXCTL.RTI</code>	0	Receive transfer initiation disable
<code>SPI_RXCTL.RWCEN</code>	0	Receive word counter disable

Table 15-11: SPI Receive Control Register (Continued)

Bits	Typical values to set	Description
SPI_RXCTL.RDR	0	Receive data request disable
SPI_RXCTL.RDO	0	Discard incoming data if RFIFO is full
SPI_RXCTL.RRWM	0	Receive FIFO regular watermark
SPI_RXCTL.RUWM	0	Receive FIFO urgent watermark disable

Table 15-12: SPI Transmit Control Register

Bits	Typical values to set	Description
SPI_TXCTL.TEN	1	Transmit channel enable
SPI_TXCTL.TTI	1	Transmit transfer initiation disable
SPI_TXCTL.TWCEN	0	Transmit word counter disable
SPI_TXCTL.TDR	0	Transmit data request disable
SPI_TXCTL.TDU	0	Send last word when TFIFO is empty
SPI_TXCTL.TRWM	0	Transmit FIFO regular watermark
SPI_TXCTL.TUWM	0	Transmit FIFO urgent watermark disable

Table 15-13: SPI DLY Control Register

Bits	Typical values to set	Description	Comments
			See Flash data sheet for CS (for example, SSEL) timing specs
SPI_DLY.LAGX	1	Extended lag timing	
SPI_DLY.LEADX	1	Extended lead timing	
SPI_DLY.STOP	3	Stop bit between the transfers	Can be set to 1 at lower SPI clock frequencies.

The multiple I/O mode (SPI_CTL.MIOM) bits are partially ignored:

- The command (opcode) is transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.CMDPINS bit setting.
- The address is then transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.ADRPINS bit setting.
- The data is always read with the number of pins specified by the SPI_CTL.MIOM bits.

NOTE: Set the SPI module enable bits SPI_CTL.EN last after configuring all registers.

Use the following programming guidelines for memory-mapped mode:

- The SPI memory-mapped hardware does not check the flash status before initiating the access. It assumes that SPI memory is always able to respond to a read access. Before enabling memory-mapped mode (for example, setting the `SPI_CTL.MMSE` bit) ensure that SPI flash is ready for a read access. When using non-memory-mapped mode, a write-complete status can be examined prior to enabling the SPI in memory-mapped mode. (See the write in progress bit in the SPI flash memory status register.) Also, immediately after initial power-up, SPI memory devices can be inaccessible for a vendor-specified period.
- When SPI is enabled in memory-mapped mode, attempts to communicate with the SPI device using legacy methods are blocked. Legacy methods include any direct access made to the transmit or receive FIFOs, whether initiated by DMA or by a processor MMR access.
- To use some of the features offered by SPI memory devices, programs can first configure the SPI memory device by setting its control word or sending some commands. Since SPI memory-mapped hardware does not allow any type of SPI write operations, configure the SPI in non-memory-mapped mode prior to enabling memory-mapped mode.
- The memory-mapped hardware does not interpret the opcode. It does not check the validity of the timing that is specified in the `SPI_MMRDH` register for a particular opcode. Programs must set the fields of the `SPI_MMRDH` register to be consistent with the read-type selected.
- When the core requests the data or code fetch, the memory-mapped transfer depends on cache settings. The cache configuration register in the SPI memory device must be appropriately configured before enabling memory-mapped mode. Some of the high performance modes like merge, wrap, and transfer size depend on cache parameters.
- SPI memory-mapped MDMA reads do not support wrapping. For MDMA reads, limit the `DMA_CFG.MSIZE` field to 1 byte, 2 bytes or 4 bytes.
- There is not always tool support to change the SPI memory-mapped hardware setting or cache settings on-the-fly. Changing these settings can optimize the performance of code that accesses SPI memory in memory-mapped mode. It is expected that the SPI memory, SPI peripheral, and cache are programmed to one specific set of control settings for the whole application. Profiling or benchmarking of the actual application can be done to find the setting that works best.

SPI Interrupt Signals

The SPI controller supports three types of interrupt request signals that correspond to data, status, and error conditions.

Data Interrupts

The SPI peripheral supports two data interrupt channels – receive and transmit. These interrupt signals are multiplexed into the DMA request lines. Since the peripheral interfaces with separate read and write interfaces with DMA, the read and write data interrupts are independent. When the DMA channels are not used, the interrupts are routed directly to the system event controller. The interrupts occupy the same vector locations as the corresponding DMA channels.

Each of the data interrupt requests can be individually controlled. Program the `SPI_RXCTL.RDR` and `SPI_TXCTL.TDR` bit fields for receive and transmit, respectively. When receive is enabled, the RX interrupt request is issued whenever there is data available in the receive datapath for reading. (The event occurs according to the `SPI_RXCTL.RDR` bit setting.) When transmit is enabled, the TX interrupt request is issued whenever the transmit datapath can be written. (The event occurs according to the `SPI_TXCTL.TDR` setting.) DMA data interrupts are compatible with second-generation DMA to incorporate urgent data requests and transfer finish interrupt requests apart from the usual data request interrupts. Transmit interrupt requests operate independently from the word counter-value in the `SPI_TWC` register.

Status Interrupts

The SPI controller supports several status interrupt requests to indicate different conditions of the receiver and transmitter. All status interrupt requests can be masked. Status interrupt requests are signaled directly through a single SPI status IRQ line. The line cannot be combined with the SPI error IRQ line for some processors. The *SPI Status Interrupts* table describes the status interrupt requests that are available for the SPI controller.

Table 15-14: SPI Status Interrupts

SPI_STAT Bit	Description
<code>SPI_STAT.RUWM</code>	Receive FIFO urgent watermark interrupt request. Issued when the level of the RFIFO breaches the watermark set in the <code>SPI_RXCTL.RUWM</code> field. It is cleared when the level of the RFIFO reaches the watermark set in the <code>SPI_RXCTL.RRWM</code> field. If the RX channel is configured in DMA mode, <code>SPI_RXCTL.RUWM</code> is multiplexed with the data request.
<code>SPI_STAT.TUWM</code>	Transmit FIFO urgent watermark interrupt request. Issued when the level of the TFIFO breaches the watermark set using the <code>SPI_TXCTL.TUWM</code> bit. It is cleared when the level of the TFIFO reaches the watermark set in the <code>SPI_TXCTL.TRWM</code> field. If the TX channel is configured in DMA mode, <code>SPI_STAT.TUWM</code> is multiplexed with the data request.
<code>SPI_STAT.TS</code>	Transmit start interrupt request. Issued when the start of a transmit burst is detected by loading of the <code>SPI_TWC</code> register with the contents of the <code>SPI_TWCR</code> register.
<code>SPI_STAT.RS</code>	Receive start interrupt request. Issued when the start of a receive burst is detected by the loading of <code>SPI_RWC</code> with the contents of <code>SPI_RWCR</code> .
<code>SPI_STAT.TF</code>	Transmit finish interrupt request. Issued when a transmit burst completes (<code>SPI_TWC</code> decrements to zero).
<code>SPI_STAT.RF</code>	Receive finish interrupt request. Issued when a receive burst completes (<code>SPI_RWC</code> decrements to zero).

Error Conditions

The SPI controller supports interrupt requests upon several different error conditions. All interrupt requests are maskable. The individual error indications combine into a single SPI error IRQ signal, which can be multiplexed on some processors with the aggregated SPI status IRQ signal. The *SPI Error Interrupts* table details the possible error indications.

Error conditions arise depending on which of the channels (transmit or receive) are enabled. If a channel is disabled, all errors related to it are ignored. When both channels are enabled, errors from both channels are enabled.

Table 15-15: SPI Error Interrupts

Bit	Description
<code>SPI_STAT.MF</code>	Mode fault. Signaled when another device also tries to be a master in a multi-master system and drives the <code>SPI_SS</code> input low. This error is signaled in master mode operation.
<code>SPI_STAT.TUR</code>	Transmission error. Signaled when an underflow condition occurs on the transmit channel. This event occurs when a new transfer starts but <code>SPI_TFIFO</code> is empty. This error does not occur in master transmit initiating mode since <code>SPI_TFIFO Not Empty</code> is one of the conditions for transfer initiation.
<code>SPI_STAT.ROR</code>	Reception error. Signaled when an overflow condition occurs on the receive channel. This event occurs when a new data word is received, but the <code>SPI_RFIFO</code> is full. This error condition does not occur in master receive initiating mode since <code>SPI_RFIFO Not Full</code> is one of the conditions for transfer initiation.
<code>SPI_STAT.TC</code>	Transmit collision error. Signaled when loading data to the transmit shift register happens near the first transmitting edge of <code>SPI_CLK</code> . In slave mode of operation, the SPI controller is unaware of when the next transfer starts. Loading of data to the transmit shift register can happen just after the transmitting edge. This event results in the setup time not being met for the first bit transmitted. The transmitted data is corrupt. In <code>SPI_CTL.CPHA 1</code> mode, the first <code>SPI_CLK</code> edge is taken as the first transmitting edge. If <code>SPI_CTL.CPHA = 0</code> , then the last <code>SPI_CLK</code> edge of the last transmission (<code>SPI_CTL.SELST = 1</code>) or slave select deassertion (<code>SPI_CTL.SELST = 0</code>) is taken as the first transmitting edge. This error is signaled only in the slave mode of operation. In master mode of operation, loading of data happens before the first transmitting edge of <code>SPI_CLK</code> .

SPI Programming Concepts

The following sections provide general programming guidelines and procedures.

Programming Guidelines

It is acceptable to program `SPI_RXCTL` and `SPI_TXCTL` registers after programming the `SPI_CTL` register. However, program the initiating mode register and its counter-register, if enabled, after the non-initiating mode register. For example, if transmit is the initiating mode and receive is the non-initiating mode, then program the `SPI_RXCTL` and `SPI_RWC` registers before the `SPI_TXCTL` and `SPI_TWC` registers. If enabling both transmit and receive in initiating mode, enable the `SPI_CTL` register after programming both the `SPI_RXCTL` and `SPI_TXCTL` registers.

These programming guidelines prevent SPI from starting a transfer when SPI registers are not fully programmed. Other ways of programming are also allowed as long as the initiating conditions prevent the start of communication until after programming of SPI registers is complete.

Avoid data corruption when changing the SPI module configuration. Do not change the configuration during a data transfer. Additionally, change the clock polarity only when no slave is selected. However, an exception to this rule exists. When an SPI communication link consists of a single master and slave, `SPI_CTL.ASSEL = 0`. The slave select input of the slave is permanently tied low. In this case, the slave is always selected. Avoid data corruption by enabling the slave only after both the master and slave devices are configured.

The module supports 8, 16-bit and 32-bit word sizes. To ensure correct operation, configure both the master and slave with the same word size.

Master Operation in Non-DMA Modes

This section describes the operation of the SPI as a master in non-DMA mode.

1. Write to the `SPI_SLVSEL` register, setting one or more of the SPI select enable bits. This operation ensures that the desired slaves are properly deselected while the master is configured.
2. The `SPI_RXCTL.RTI` and `SPI_TXCTL.TTI` bits determine the SPI initiating mode. The initiating mode defines the primary transfer channel, and also the initiating condition for the transfer.
3. Write to the `SPI_CLK`, `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers. This operation enables the device as a master and configures the SPI system. It specifies the transfer modes and channels, appropriate word length, transfer format, baud rate, and other control information.

ADDITIONAL INFORMATION: If `SPI_RXCTL.RTI` is enabled and `SPI_TXCTL.TTI` is not, write to the `SPI_RXCTL` register after writing into `SPI_CTL`, `SPI_TXCTL`, and `SPI_TFIFO` registers to prevent a transmit underrun for the first transfer.

4. If `SPI_CTL.ASSEL=0`, activate the desired slaves by clearing one or more of the `SPI_SLVSEL` flag bits. Otherwise, the SPI hardware performs slave activation.
5. The SPI controller then generates the programmed clock pulses on `SPI_CLK` and simultaneously shifts data out of `SPI_MOSI` while shifting data in from `SPI_MISO`. Before a shift, the shift register is loaded with the contents of the `SPI_TFIFO` register. At the end of the transfer, the contents of the shift register are loaded into `SPI_RFIFO`.
6. Whenever the initiating conditions are satisfied, the SPI continues to send and receive words. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the `SPI_TXCTL.TDU` and `SPI_RXCTL.RDO` bits.
7. It is possible to program a secondary channel in addition to the initiating channel. This feature allows usage of available channel resources for receives or transmits simultaneously with the initiating channel.

Slave Operation in Non-DMA Modes

When a device is enabled as a slave in a non-DMA mode, a transition of the `SPI_SS` select signal to the active state (low) triggers the the start of a transfer. Or, the first active edge of `SPI_CLK` triggers the start, depending on the state of `SPI_CTL.CPHA` bit. The interface operates in the following manner.

1. The core writes to the `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers. The operation defines the mode of the serial link to be the same as the mode setup in the SPI master.
2. To prepare for the data transfer, the core writes data to be transmitted into `SPI_TFIFO`.
3. Once the `SPI_SS` falling edge is detected, the slave starts sending data on active `SPI_CLK` edges and sampling data on inactive `SPI_CLK` edges.
4. Reception or transmission continues until `SPI_SS` is released or until the slave has received the proper number of clock cycles.

- The slave device continues to receive or transmit with each new falling edge transition on $\overline{\text{SPI_SS}}$ or active SPI_CLK edge. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the SPI_TXCTL.TDU and SPI_RXCTL.RDO bits.

Configuring DMA Master Mode

The SPI interface supports a write DMA channel and a read DMA channel. It can use these functions individually or in a lock-step manner in duplex mode ($\text{SPI_TXCTL.TTI} = \text{SPI_RXCTL.RTI} = 1$).

- Write to the appropriate DMA registers to enable the SPI DMA channel and to configure the necessary work units, access direction, word count, and so on.
- Write to the SPI_SLVSEL register, setting one or more of the SPI flag select bits.
- Write to the SPI_CLK and SPI_CTL registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and so forth.
- Write to SPI_RXCTL to configure SPI master receive mode, or write to SPI_TXCTL to configure SPI master transmit mode.
- Finally, write to the SPI_RXCTL.REN bit to enable the receive channel, or write to SPI_TXCTL.TEN to enable the transmit channel.
- If the SPI_RXCTL.RTI bit is enabled, a receive transfer is initiated upon enabling SPI_CTL.EN bit. If the receive word counter is enabled (SPI_RXCTL.RWCEN), then the SPI_RWC register must be non-zero for a transfer to initiate.

ADDITIONAL INFORMATION: If enabling both receive and transmit DMA channels, but not enabling SPI_TXCTL.TTI , write to the SPI_RXCTL register after writing the SPI_CTL and SPI_TXCTL registers. In this way, a transmit underrun can be prevented for the first transfer. Subsequent transfers are initiated as the SPI reads data from the receive shift register and writes to the SPI receive FIFO. The SPI then requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the SPI receive FIFO and writes to memory. New requests continue to be initiated as long as the receive FIFO does not fill up, when SPI_RWC does not become zero while $\text{SPI_RXCTL.RWCEN} = 1$.

- If SPI_TXCTL.TTI is enabled, the SPI controller requests DMA reads from memory as long as there is space for more data in the transmit pipe. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO. As long as transmit data is available in the FIFO, and the SPI_TWC register is non-zero when $\text{SPI_TXCTL.TWCEN} = 1$, the SPI continues to initiate transfers until disabled.
- If both the SPI_TXCTL.TTI and SPI_RXCTL.RTI bits are enabled, the SPI controller requests a DMA read from memory. However, there must be space for more data in the transmit pipe and the number of words written into the SPI must be less than SPI_TWC if $\text{SPI_TXCTL.TWCEN} = 1$. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.

ADDITIONAL INFORMATION: As the SPI writes data from the transmit FIFO into the transmit shift register, it initiates a transfer on the SPI link. Data received from the transfer is moved from the SPI receive shift register to the receive FIFO. The SPI controller requests a write from DMA to memory. Upon a DMA grant,

the DMA engine reads a word from the receive FIFO and writes to memory. Transfer continues to be initiated as long as both receives and transmits can accommodate new data.

9. If the receive pipe fills up due to unavailability of DMA grants, the transmit pipe stalls until the pipe is drained. If the transmit pipe fills up, the SPI stops requesting for DMA writes. If the value in `SPI_RWC` expires, further write-requests to DMA stop. However, data already written into the transmit FIFO is sent, and read requests to DMA continue until the receive data is read from the receive FIFO.
10. The SPI then generates the programmed clock pulses on `SPI_CLK` and simultaneously shifts data out of `SPI_MOSI` while shifting data in from `SPI_MISO`. For receive transfers, the value in the shift register is loaded into the `SPI_RFIFO` register at the end of the transfer. For transmit transfers, the value in the `SPI_TFIFO` register is loaded into the shift register at the start of the transfer.

Configuring DMA Slave Mode Operation

This mode occurs when the SPI is enabled as a slave and the DMA engine is configured to transmit or receive data. A transition of the `SPI_SS` signal to the active-low state triggers the start of a transfer. Or, the first active edge of `SPI_CLK` triggers the start of a transfer, depending on the state of the `SPI_CTL.CPHA` bit. The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave (in response to a master command). The SPI supports a receive DMA channel and a transmit DMA channel.

1. Write to the appropriate DMA registers to enable the SPI DMA channel and configure the necessary work units, access direction, word count, and so on.
2. Write to the `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers to define the mode of the serial link to be the same as the mode configured in the SPI master.
3. If the receive channel is enabled (`SPI_RXCTL.REN` is asserted), the following actions occur:
 - a. Once the slave select input is active, the slave starts receiving and transmitting data on active `SPI_CLK` edges.
 - b. The value in the shift register is loaded into the `SPI_RFIFO` register at the end of the transfer.
 - c. Once `SPI_RFIFO` has valid data, it requests a write from DMA to memory.
 - d. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory.
 - e. As long as there is data in the receive FIFO, the SPI slave continues to request a DMA write to memory. The DMA engine continues to read a word from the FIFO and writes to memory. The SPI slave continues receiving words on active `SPI_CLK` edges as long as the `SPI_SS` input is active.
 - f. If the data collected in the receive pipe breaches the set level, and the DMA engine cannot keep up with the receive rate, the slave can deassert the `SPI_RDY` signal. This signaling throttles the master. The receive pipe level is set according to the `SPI_CTL.FCWM` field. The signal is deasserted as the DMA drains the receive FIFO. Alternatively, the SPI can use the `SPI_RXCTL.RDO` bit to decide when the incoming data is discarded or overwritten into the receive FIFO (when `SPI_CTL.FCEN` is inactive).
4. If the transmit channel is enabled (`SPI_TXCTL.TEN` is asserted), the following actions occur:

- a. The SPI requests a DMA read from memory.
 - b. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.
 - c. The SPI then reads DMA data from the transmit FIFO and writes to the transmit shift register, awaiting the start of the next transfer.
 - d. Once the slave select input is active, the slave starts receiving and transmitting data on active `SPI_CLK` edges.
 - e. As long as there is room in the transmit FIFO, the SPI slave continues to request a DMA read from memory. The DMA engine continues to read a word from memory and write to the transmit FIFO. The SPI slave continues transmitting words on active `SPI_CLK` edges as long as the `SPI_SS` input is active.
 - f. If the number of outstanding data entries in the transmit pipe breaches the level set and the DMA cannot keep up with the transmit rate, the slave deasserts the `SPI_RDY` signal. This signaling throttles the master. The transmit pipe level is set according to the `SPI_CTL.FCWM` field. The signal is deasserted as the DMA fills the transmit FIFO. Alternately, the `SPI_TXCTL.TDU` bit decides the state of the transmit data (when `SPI_CTL.FCEN` is deasserted).
5. If both receive and transmit channels are enabled, the following actions occur after the actions for each channel. Transfers continue as long as both receive and transmit channels can accommodate new data.
- a. If the receive pipe fills up due to the unavailability of DMA grant, the SPI interface stalls the master by asserting the `SPI_RDY` pin. This signal is deasserted as the DMA drains the receive FIFO. Alternately, the SPI uses the `SPI_RXCTL.RDO` bit to decide when the incoming data is discarded or overwritten in the receive FIFO (when `SPI_CTL.FCEN` is deasserted).
 - b. If the transmit pipe fills up, the SPI stops requesting DMA writes until the pipe clears.
 - c. If there is an underflow problem in the transmit pipe, the slave stalls the master by deasserting `SPI_RDY` while the DMA fills the transmit FIFO. Alternately, the SPI uses the `SPI_TXCTL.TDU` bit to decide the state of the transmit data (when `SPI_CTL.FCEN` is deasserted).

ADSP-2156x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 15-16: ADSP-2156x SPI Register List

Name	Description
<code>SPI_CLK</code>	Clock Rate Register
<code>SPI_CTL</code>	Control Register
<code>SPI_DLY</code>	Delay Register
<code>SPI_ILAT</code>	Masked Interrupt Condition Register
<code>SPI_ILAT_CLR</code>	Masked Interrupt Clear Register

Table 15-16: ADSP-2156x SPI Register List (Continued)

Name	Description
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

Clock Rate Register

The `SPI_CLK` register selects the baud rate for SPI data transfers, relating this rate to the SPI serial clock (SPI clock) and the system clock (SCLK).

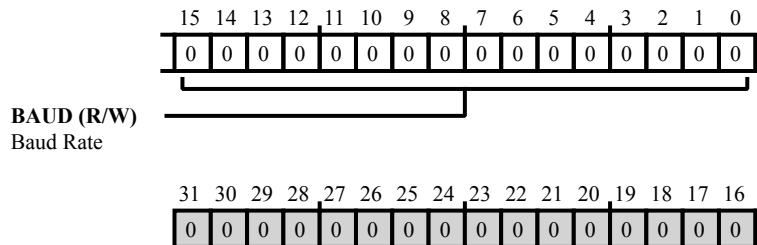


Figure 15-20: SPI_CLK Register Diagram

Table 15-17: SPI_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	BAUD	Baud Rate. The <code>SPI_CLK.BAUD</code> bits set the SPI baud rate according to the formula: $BAUD = (SCLK / SPI\ Clock) - 1$

Control Register

The `SPI_CTL` register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

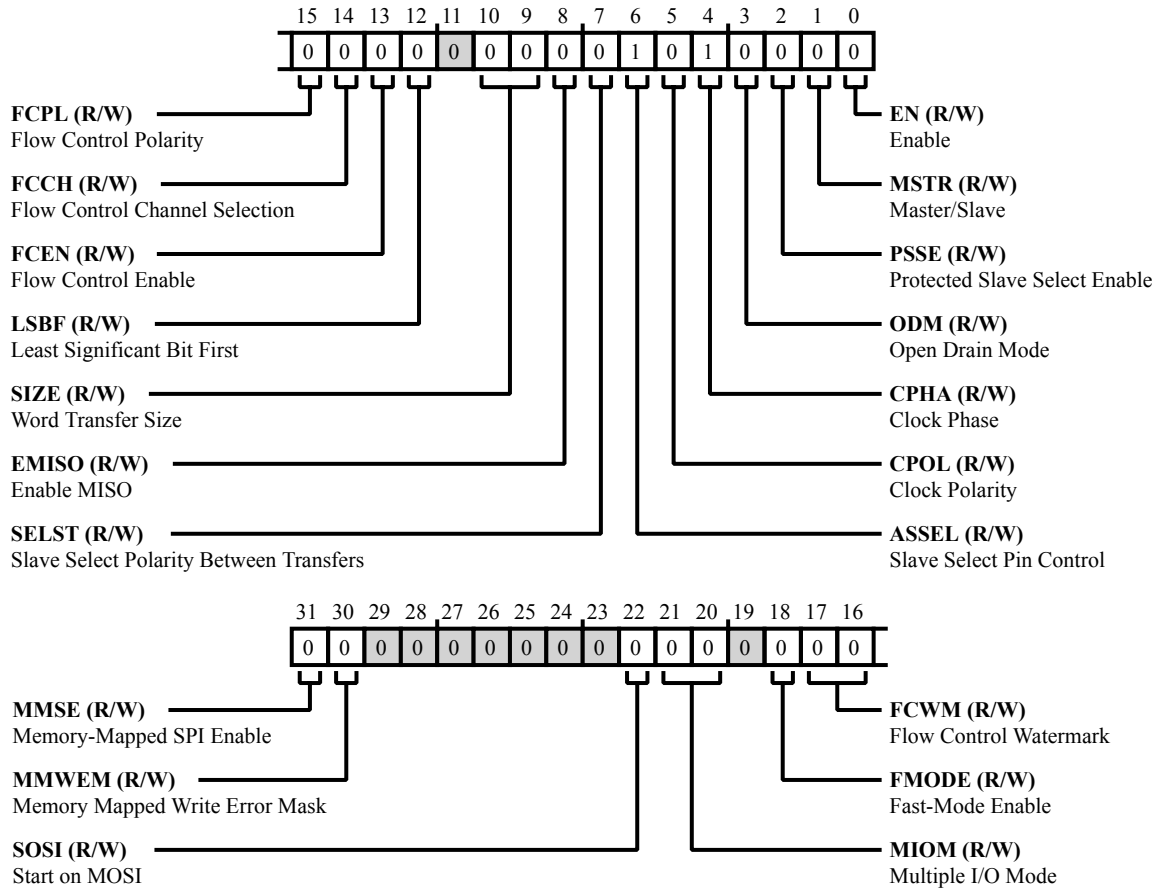


Figure 15-21: SPI_CTL Register Diagram

Table 15-18: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MMSE	Memory-Mapped SPI Enable. When the <code>SPI_CTL.MMSE</code> bit is asserted, communication to an SPI memory device is automated such that the memory it contains is accessible directly through the read of processor address space assigned to it. (As far as the SPI peripheral is concerned, this includes all read accesses received by the SPI peripherals system crossbar slave port.) Note that when memory-mapped access of SPI memory is enabled, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses are generated. Legacy methods include any direct access made to the Tx and Rx FIFOs, whether initiated by DMA or processor MMR access.
		0 Hardware automated access of memory-mapped SPI memory disabled.
		1 Hardware-automated access of memory-mapped SPI memory enabled.
30 (R/W)	MMWEM	Memory Mapped Write Error Mask. The <code>SPI_CTL.MMWEM</code> bit specifies whether an error response is returned to the fabric upon write attempts to address space reserved for memory-mapped reads of SPI memory.
		0 Write error response returned upon write attempts to memory-mapped SPI memory
		1 Write error response masked (not returned) upon write attempts to memory-mapped SPI memory
22 (R/W)	SOSI	Start on MOSI. The <code>SPI_CTL.SOSI</code> bit is valid only when <code>SPI_CTL.MIOM</code> is enabled for either DIOM or QIOM, and this bit selects the starting pin and the bit placement on pins for these modes. In DIOM, by default, (<code>SPI_CTL.SOSI = 0</code>) SPI sends the first bit on the <code>SPI_MISO</code> pin and the second bit on the <code>SPI_MOSI</code> pin. In QIOM, by default, the SPI sends the first bit on the <code>SPI_D3</code> pin, the second bit on the <code>SPI_D2</code> pin, the third bit on the <code>SPI_MISO</code> pin and the fourth bit on the <code>SPI_MOSI</code> pin. This order can be reversed by setting the <code>SPI_CTL.SOSI</code> bit. When this bit is set, the SPI sends the first bit on the <code>SPI_MOSI</code> pin. The first bit referred to here depends on the <code>SPI_CTL.LSBF</code> bit setting (MSB bit or LSB bit).
		0 Start on MISO (DIOM) or start on SPI_D3
		1 Start on MOSI

Table 15-18: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21:20 (R/W)	MIOM	Multiple I/O Mode. The <code>SPI_CTL.MIOM</code> bits enable SPI operation in dual I/O mode (DIOM) or quad I/O mode (QIOM). These bits can only be changed when the SPI is disabled (<code>SPI_CTL.EN = 0</code>).
		0 No MIOM (disabled)
		1 DIOM operation
		2 QIOM operation
		3 Reserved
18 (R/W)	FMODE	Fast-Mode Enable. The <code>SPI_CTL.FMODE</code> bit enables fast mode operation for SPI receive transfers. SPI transmit operations in fast mode are the same as normal mode.
		0 Disable
		1 Enable
17:16 (R/W)	FCWM	Flow Control Watermark. The <code>SPI_CTL.FCWM</code> bits select the watermark level of the transmit channel (<code>SPI_TFIFO</code> buffer) or receive channel (<code>SPI_RFIFO</code> buffer) that triggers flow control operation. These bits are applicable only when the SPI is a slave (<code>SPI_CTL.MSTR = 0</code>) and flow control is enabled (<code>SPI_CTL.FCEN = 1</code>). When the watermark condition is met, the SPI slave deasserts the <code>SPI_RDY</code> pin.
		0 TFIFO empty or RFIFO full
		1 TFIFO 75% or more empty, or RFIFO 75% or more full
		2 TFIFO 50% or more empty, or RFIFO 50% or more full
		3 Reserved
15 (R/W)	FCPL	Flow Control Polarity. The <code>SPI_CTL.FCPL</code> bit selects flow control polarity for the <code>SPI_RDY</code> pin when flow control is enabled. When the <code>SPI_RDY</code> pin is active, the SPI is indicating it is ready for data transfer.
		0 Active-low RDY
		1 Active-high RDY

Table 15-18: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	FCCH	Flow Control Channel Selection. The <code>SPI_CTL.FCCH</code> bit selects whether the SPI applies flow control to the transmit channel (<code>SPI_TFIFO</code> buffer) or receive channel (<code>SPI_RFIFO</code> buffer). This bit is applicable only when the SPI is a slave and flow control is enabled.
		0 Flow control on RX buffer
		1 Flow control on TX buffer
13 (R/W)	FCEN	Flow Control Enable. The <code>SPI_CTL.FCEN</code> bit enables SPI flow control operation, which permits slow slave devices to interface with fast master devices. This bit controls the operation of the <code>SPI_RDY</code> pin. Note that options for flow control operation are available using the <code>SPI_CTL.FCCH</code> , <code>SPI_CTL.FCPL</code> , and <code>SPI_CTL.FCWM</code> bits.
		0 Disable
		1 Enable
12 (R/W)	LSBF	Least Significant Bit First. The <code>SPI_CTL.LSBF</code> bit selects whether the SPI transmits/receives data as LSB first (little endian) or MSB first (big endian). This bit can only be changed when the SPI is disabled.
		0 MSB sent/received first (big endian)
		1 LSB sent/received first (little endian)
10:9 (R/W)	SIZE	Word Transfer Size. The <code>SPI_CTL.SIZE</code> bits select the SPI transfer word size as 8, 16 or 32 bits. To ensure correct operation, both the master and slave must be configured with the same word size. This bit can only be changed when the SPI is disabled (<code>SPI_CTL.EN=0</code>).
		0 8-bit word
		1 16-bit word
		2 32-bit word
		3 Reserved
8 (R/W)	EMISO	Enable MISO. The <code>SPI_CTL.EMISO</code> bit enables master-in-slave-out (MISO) mode. This SPI mode is applicable only when the SPI is a slave.
		0 Disable
		1 Enable

Table 15-18: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The <code>SPI_CTL.SELST</code> bit selects the state (polarity) for the <code>SPI_SEL[n]</code> pin between SPI transfers when the SPI is a master and hardware slave select assertion is enabled (<code>SPI_CTL.ASSEL=1</code>). In slave mode, this bit affects the detection of both transmit collision (<code>SPI_STAT.TC</code> and underrun (<code>SPI_STAT.TUR</code>) errors.
		0 Deassert slave select (high)
		1 Assert slave select (low)
6 (R/W)	ASSEL	Slave Select Pin Control. The <code>SPI_CTL.ASSEL</code> bit selects whether the SPI hardware sets the <code>SPI_SEL[n]</code> pin output value (ignoring the slave select <code>SPI_SLVSEL.SSEL1 - SPI_SLVSEL.SSEL7</code> bits) or whether software control of the slave select bits set the <code>SPI_SEL[n]</code> pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the <code>SPI_SEL[n]</code> pin output is asserted during the transfers, and the pin polarity between transfers is selected by the <code>SPI_CTL.SELST</code> bit. When software control is enabled, the <code>SPI_SEL[n]</code> pin output value is set through software control of the slave select bits, and as such, the pin may either remain asserted (low) or be deasserted between transfers.
		0 Software slave select control
		1 Hardware slave select control
5 (R/W)	CPOL	Clock Polarity. The <code>SPI_CTL.CPOL</code> bit selects whether the SPI uses an active-low or active-high signal for the SPI clock (<code>SPI_CLK</code>). This bit works with the <code>SPI_CTL.CPHA</code> bit to select combinations of clock phase and polarity for the <code>SPI_CLK</code> pin. This bit can only be changed when the SPI is disabled.
		0 Active-high SPI CLK
		1 Active-low SPI CLK
4 (R/W)	CPHA	Clock Phase. The <code>SPI_CTL.CPHA</code> bit selects whether the SPI starts toggling the signal for the SPI clock (<code>SPI_CLK</code>) from the start of the first data bit or from the middle of the first data bit. The <code>SPI_CTL.CPHA</code> bit works with the <code>SPI_CTL.CPOL</code> bit to select combinations of clock phase and polarity for the <code>SPI_CLK</code> pin. This bit can only be changed when the SPI is disabled.
		0 SPI CLK toggles from middle
		1 SPI CLK toggles from start

Table 15-18: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	ODM	Open Drain Mode. The <code>SPI_CTL.ODM</code> bit configures the data output pins (<code>SPI_MOSI</code> and <code>SPI_MISO</code>) to behave as open drain outputs, which prevents contention and possible damage to pin drivers in multi-master or multi-slave SPI systems. When <code>SPI_CTL.ODM</code> is enabled and the SPI is a master, the SPI three-states the <code>SPI_MOSI</code> pin when the data driven out on <code>MOSI</code> is a logic-high. The SPI does not three-state the <code>SPI_MOSI</code> pin when the driven data is a logic-low. When <code>SPI_CTL.ODM</code> is enabled and the SPI is a slave, the SPI three-states the <code>SPI_MISO</code> pin when the data driven out on <code>SPI_MISO</code> is a logic-high. Note that an external pull-up resistor is required on both the <code>SPI_MOSI</code> and <code>SPI_MISO</code> pins when <code>SPI_CTL.ODM</code> is enabled.
		0 Disable
		1 Enable
2 (R/W)	PSSE	Protected Slave Select Enable. The <code>SPI_CTL.PSSE</code> bit enables the <code>SPI_SS</code> pin to provide error detection input in a multi-master environment when the SPI is in master mode. If some other device in the system asserts the <code>SPI_SS</code> pin while SPI is enabled as master (and <code>SPI_CTL.PSSE</code> is enabled), this condition causes a mode fault error.
		0 Disable
		1 Enable
1 (R/W)	MSTR	Master/Slave. The <code>SPI_CTL.MSTR</code> bit toggles the SPI between master mode and slave mode. This bit can only be changed when the SPI is disabled.
		0 Slave
		1 Master
0 (R/W)	EN	Enable. The <code>SPI_CTL.EN</code> bit enables SPI operation.
		0 Disable SPI module
		1 Enable

Delay Register

The `SPI_DLY` register selects a transfer delay and the lead/lag timing between slave select signals and SPI clock edge assertion/deassertion.

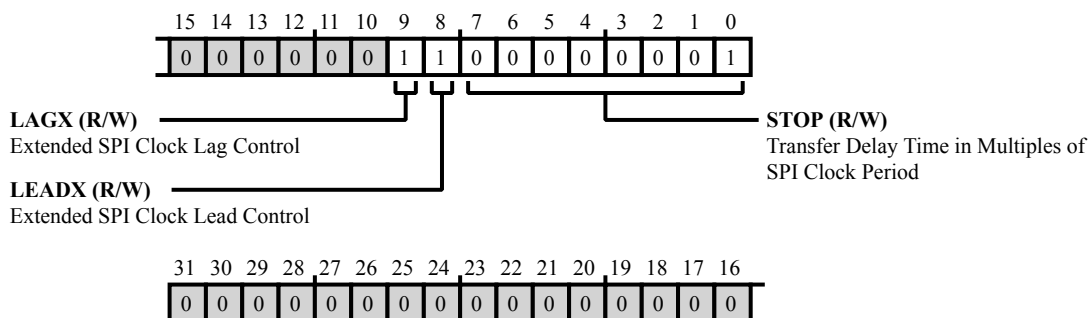


Figure 15-22: `SPI_DLY` Register Diagram

Table 15-19: `SPI_DLY` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	LAGX	Extended SPI Clock Lag Control. The <code>SPI_DLY.LAGX</code> bit enables insertion of a 1- <code>SPI_CLK</code> cycle lag (extend lag) in the timing between the slave select (<code>SPI_SEL[n]</code>) assertion and first SPI clock edge.
		0 Disable
		1 Enable
8 (R/W)	LEADX	Extended SPI Clock Lead Control. The <code>SPI_DLY.LEADX</code> bit enables insertion of a 1- <code>SPI_CLK</code> cycle lead (extend lead) in the timing between the slave select (<code>SPI_SEL[n]</code>) deassertion and last SPI clock edge.
		0 Disable
		1 Enable
7:0 (R/W)	STOP	Transfer Delay Time in Multiples of SPI Clock Period. The <code>SPI_DLY.STOP</code> bits select a delay (number of stop bits in multiples of SPI clock duration) at the end of each SPI transfer. The default delay is the minimum value required to comply with the SPI protocol (1-bit duration). The <code>SPI_DLY.STOP</code> bits can be programmed with smaller delay values, resulting in continuous operation (for example, stop bits =0).

Masked Interrupt Condition Register

The `SPI_ILAT` register latches interrupts, queuing the interrupt requests for service. When a condition is indicated by a bit in the `SPI_STAT` register and the corresponding interrupt request is unmasked in `SPI_IMSK`, the SPI latches the interrupt request bit in `SPI_ILAT`.

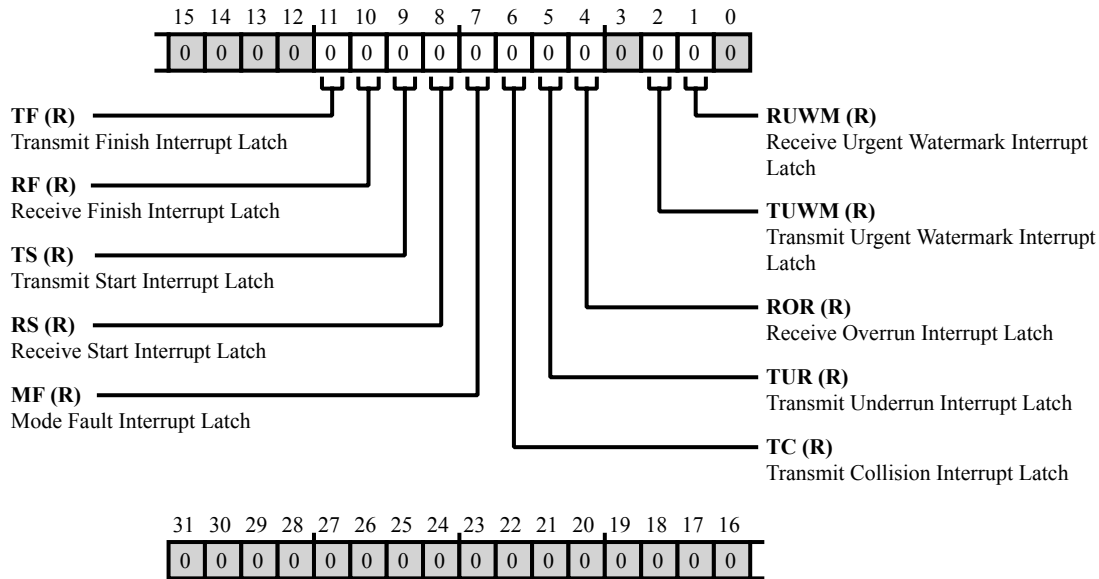


Figure 15-23: SPI_ILAT Register Diagram

Table 15-20: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	TF	Transmit Finish Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
10 (R/NW)	RF	Receive Finish Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
9 (R/NW)	TS	Transmit Start Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
8 (R/NW)	RS	Receive Start Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request

Table 15-20: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	MF	Mode Fault Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
6 (R/NW)	TC	Transmit Collision Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
5 (R/NW)	TUR	Transmit Underrun Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.
		0 No interrupt request
		1 Latched interrupt request

Masked Interrupt Clear Register

The `SPI_ILAT_CLR` register permits clearing individual mask bits in the `SPI_ILAT` register without affecting other bits in the register. Use write-1-to-clear on a bit in the `SPI_ILAT_CLR` register to clear the corresponding bit in the `SPI_ILAT` register.

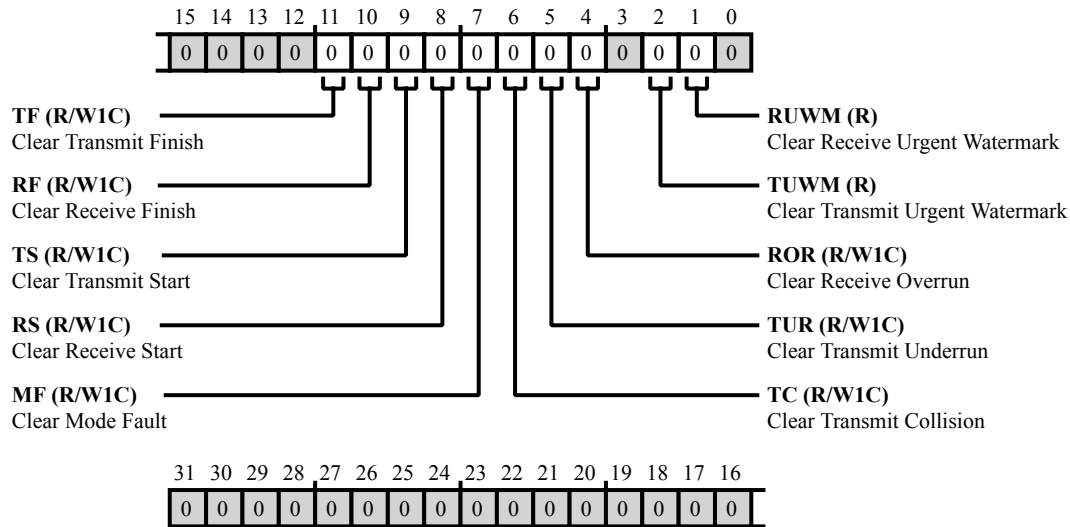


Figure 15-24: `SPI_ILAT_CLR` Register Diagram

Table 15-21: `SPI_ILAT_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1C)	TF	Clear Transmit Finish. The <code>SPI_ILAT_CLR.TF</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The <code>SPI_ILAT_CLR.RF</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The <code>SPI_ILAT_CLR.TS</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit

Table 15-21: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	RS	Clear Receive Start. The <code>SPI_ILAT_CLR.RS</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The <code>SPI_ILAT_CLR.MF</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The <code>SPI_ILAT_CLR.TC</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The <code>SPI_ILAT_CLR.TUR</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The <code>SPI_ILAT_CLR.ROR</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit
2 (R/NW)	TUWM	Clear Transmit Urgent Watermark. The <code>SPI_ILAT_CLR.TUWM</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.
		0 No effect
		1 Clear mask bit

Table 15-21: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	RUWM	Clear Receive Urgent Watermark. The <code>SPI_ILAT_CLR.RUWM</code> bit clears the corresponding mask bit in the <code>SPI_ILAT</code> register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Register

The `SPI_IMSK` register unmask (enables) or mask (disables) SPI interrupt requests. When a condition is indicated by a bit in the `SPI_STAT` register and the corresponding interrupt request is unmasked in `SPI_IMSK`, the SPI latches the interrupt request bit in the `SPI_ILAT` register, queuing the interrupt request for service.

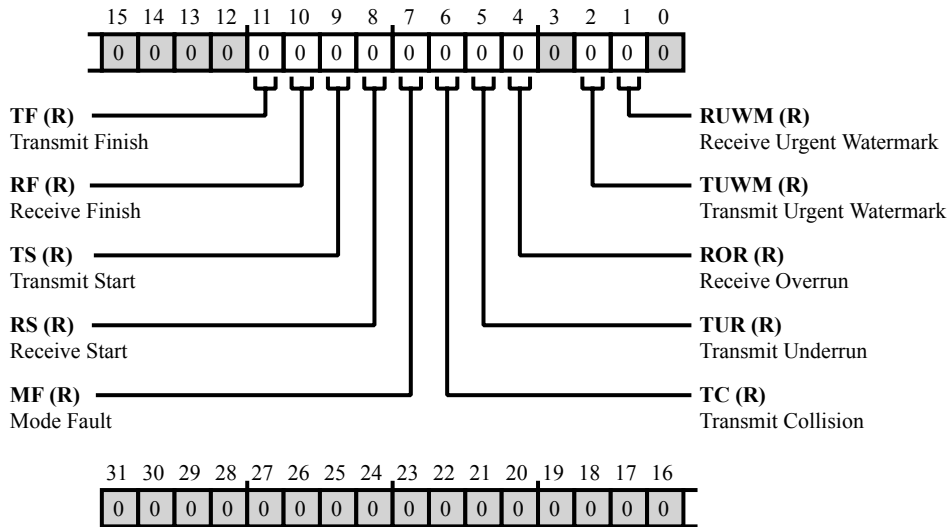


Figure 15-25: SPI_IMSK Register Diagram

Table 15-22: SPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	TF	Transmit Finish. The <code>SPI_IMSK.TF</code> bit unmask (enables) or mask (disables) the TF interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
10 (R/NW)	RF	Receive Finish. The <code>SPI_IMSK.RF</code> bit unmask (enables) or mask (disables) the RF interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
9 (R/NW)	TS	Transmit Start. The <code>SPI_IMSK.TS</code> bit unmask (enables) or mask (disables) the TS interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request

Table 15-22: SPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/NW)	RS	Receive Start. The <code>SPI_IMSK.RS</code> bit unmask (enables) or mask (disables) the RS interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
7 (R/NW)	MF	Mode Fault. The <code>SPI_IMSK.MF</code> bit unmask (enables) or mask (disables) the MF interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
6 (R/NW)	TC	Transmit Collision. The <code>SPI_IMSK.TC</code> bit unmask (enables) or mask (disables) the TC interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
5 (R/NW)	TUR	Transmit Underrun. The <code>SPI_IMSK.TUR</code> bit unmask (enables) or mask (disables) the TUR interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
4 (R/NW)	ROR	Receive Overrun. The <code>SPI_IMSK.ROR</code> bit unmask (enables) or mask (disables) the ROR interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark. The <code>SPI_IMSK.TUWM</code> bit unmask (enables) or mask (disables) the TUWM interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark. The <code>SPI_IMSK.RUWM</code> bit unmask (enables) or mask (disables) the RUWM interrupt.
		0 Disable (mask) interrupt request
		1 Enable (unmask) interrupt request

Interrupt Mask Clear Register

The `SPI_IMSK_CLR` register permits clearing individual mask bits in the `SPI_IMSK` register without affecting other bits in the register. Use write-1-to-clear on a bit in the `SPI_IMSK_CLR` register to clear the corresponding bit in the `SPI_IMSK` register.

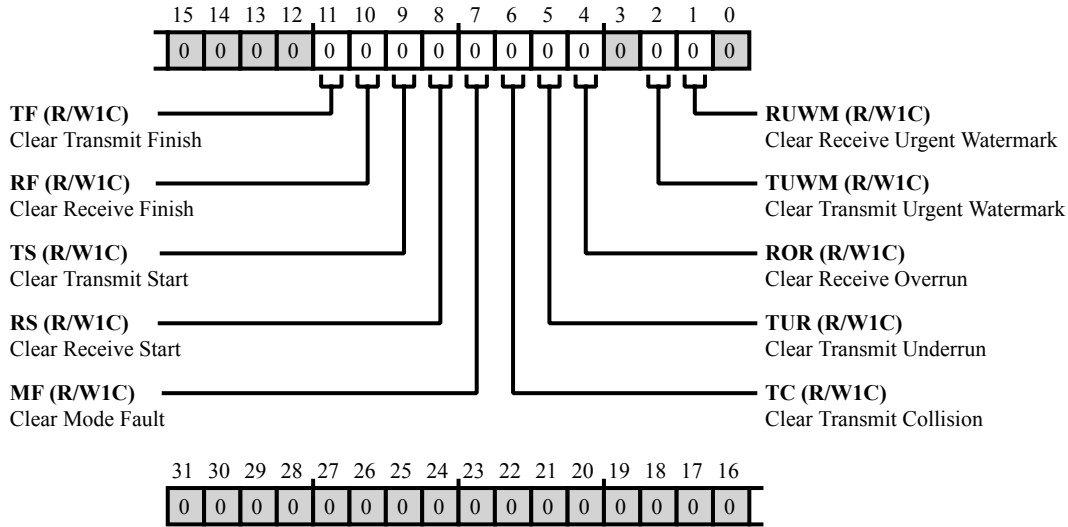


Figure 15-26: `SPI_IMSK_CLR` Register Diagram

Table 15-23: `SPI_IMSK_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1C)	TF	Clear Transmit Finish. The <code>SPI_IMSK_CLR.TF</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The <code>SPI_IMSK_CLR.RF</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The <code>SPI_IMSK_CLR.TS</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit

Table 15-23: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	RS	Clear Receive Start. The <code>SPI_IMSK_CLR.RS</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The <code>SPI_IMSK_CLR.MF</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The <code>SPI_IMSK_CLR.TC</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The <code>SPI_IMSK_CLR.TUR</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The <code>SPI_IMSK_CLR.ROR</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit
2 (R/W1C)	TUWM	Clear Transmit Urgent Watermark. The <code>SPI_IMSK_CLR.TUWM</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Clear mask bit

Table 15-23: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RUWM	Clear Receive Urgent Watermark. The <code>SPI_IMSK_CLR.RUWM</code> bit clears the corresponding mask bit in the <code>SPI_IMSK</code> register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Set Register

The `SPI_IMSK_SET` register permits setting individual mask bits in the `SPI_IMSK` register without affecting other bits in the register. Use write-1-to-set on a bit in the `SPI_IMSK_SET` register to set the corresponding bit in the `SPI_IMSK` register.

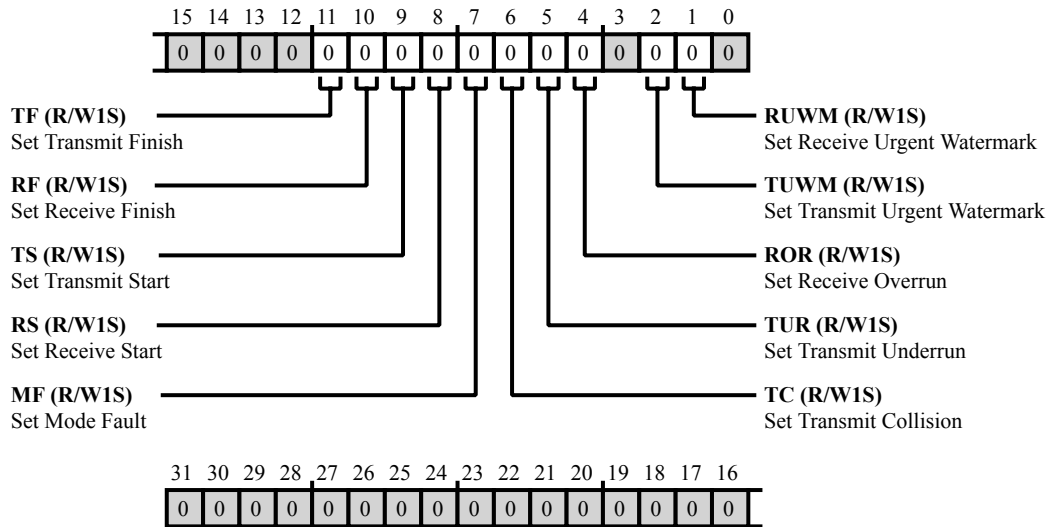


Figure 15-27: SPI_IMSK_SET Register Diagram

Table 15-24: SPI_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1S)	TF	Set Transmit Finish. The <code>SPI_IMSK_SET.TF</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
10 (R/W1S)	RF	Set Receive Finish. The <code>SPI_IMSK_SET.RF</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
9 (R/W1S)	TS	Set Transmit Start. The <code>SPI_IMSK_SET.TS</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit

Table 15-24: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1S)	RS	Set Receive Start. The <code>SPI_IMSK_SET.RS</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
7 (R/W1S)	MF	Set Mode Fault. The <code>SPI_IMSK_SET.MF</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
6 (R/W1S)	TC	Set Transmit Collision. The <code>SPI_IMSK_SET.TC</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
5 (R/W1S)	TUR	Set Transmit Underrun. The <code>SPI_IMSK_SET.TUR</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
4 (R/W1S)	ROR	Set Receive Overrun. The <code>SPI_IMSK_SET.ROR</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit
2 (R/W1S)	TUWM	Set Transmit Urgent Watermark. The <code>SPI_IMSK_SET.TUWM</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit

Table 15-24: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1S)	RUWM	Set Receive Urgent Watermark. The <code>SPI_IMSK_SET.RUWM</code> bit sets the corresponding mask bit in the <code>SPI_IMSK</code> register.
		0 No effect
		1 Set mask bit

Memory Mapped Read Header

The `SPI_MMRDH` register enables the use of memory-mapped mode. This mode allows direct memory-mapped read accesses of an SPI memory device and is primarily used to directly execute instructions from an SPI FLASH memory without using a low-level software driver. All overhead tasks such as transmission of the read header, pin turnaround timing and receive data sizing are handled in hardware.

The memory-mapped access mode is enabled by setting the `SPI_CTL.MMSE` bit. The features within the `SPI_MMRDH` register include a command skip mode, variable length byte addressing, and independent multi-pin support for command transmission, address transmission and data reception. In addition, the command opcode and mode bytes are fully programmable.

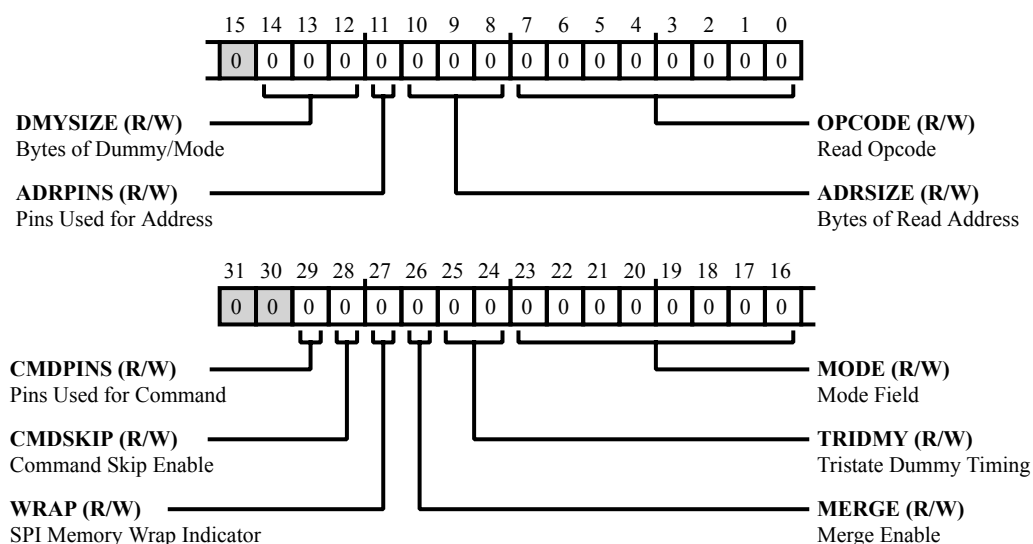


Figure 15-28: `SPI_MMRDH` Register Diagram

Table 15-25: `SPI_MMRDH` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	CMDPINS	Pins Used for Command. The <code>SPI_MMRDH.CMDPINS</code> bit specifies the number of pins to be used for command transmission. This bit must be set consistent with the expectations established by the read opcode. Hardware does not interpret <code>SPI_MMRDH.OPCODE</code> , but rather relies on this bit to specify behavior. When cleared, it overrides the <code>SPI_CTL.MIOM</code> bits. When set, it uses bits specified by the <code>SPI_CTL.MIOM</code> bit setting.
		0 Use only one pin: MOSI (overrides <code>SPI_CTL.MIOM</code> bits)
		1 Use pins specified by <code>SPI_CTL.MIOM</code> bits

Table 15-25: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	CMDSKIP	Command Skip Enable. The <code>SPI_MMRDH.CMDSKIP</code> bit enables command skip mode where the address is sent first and the <code>OPCODE</code> field is not sent (<code>SPI_MMRDH.CMDSKIP</code> bit =1). This mode is useful for supporting XIP (Execute-In-Place) operation where only the address is sent and the same read command is assumed. The SPI flash device must be primed with an initial read command, before the <code>SPI_MMRDH.CMDSKIP</code> bit is set.
		0 <code>OPCODE</code> field is sent first followed by address
		1 <code>OPCODE</code> field is not sent; address is sent first
27 (R/W)	WRAP	SPI Memory Wrap Indicator. The <code>SPI_MMRDH.WRAP</code> bit must be set by software if software places a connected SPI memory device into a 8-byte, 16-byte or 32-byte wrap mode based on the <code>ILINE</code> and <code>DLINE</code> field setting of the cache configuration register address wrap mode. Software achieves this by transmitting a vendor specified command to the SPI memory device while the <code>SPI_CTL.MMSE</code> bit =0. If the <code>SPI_MMRDH.WRAP</code> bit =1, the SPI does not need to deassert the SPI slave select signal and resend the read header in order to wrap to the cache line base when servicing misaligned cache fill requests. Although this improves cache fill efficiency, it requires that the SPI deassert the SPI slave select pin and resend the read header whenever a DMA burst requests crosses 32 byte alignments. Setting this bit improves cache throughput but decreases DMA throughput.
		0 SPI Memory auto increments address purely sequentially
		1 SPI Memory auto increments address but wraps within 32 Byte lines
26 (R/W)	MERGE	Merge Enable. When the <code>SPI_MMRDH.MERGE</code> bit is set, SPI hardware combines the two successive transfers. This increases the throughput rate when accessing a large number of sequential memory locations. For more information refer to the Merged Read Accesses section.

Table 15-25: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/W)	TRIDMY	Tristate Dummy Timing. The <code>SPI_MMRDH.TRIDMY</code> bits specify whether and when output pins are three-stated during the interval of time specified by the <code>SPI_MMRDH.DMYSIZE</code> bits. Output pins potentially three-stated include all pins which were used to transmit the address
		0 Tristate outputs immediately
		1 Tristate outputs after 4 bits of dummy/mode are transmitted
		2 Tristate outputs after 8 bits of dummy/mode are transmitted
		3 Never tristate outputs (previously specified output state is held)
23:16 (R/W)	MODE	Mode Field. These bits specify up to a leading byte to be transmitted during the interval of time specified by the <code>SPI_MMRDH.DMYSIZE</code> bit field. This first byte, or a portion of it, is interpreted as mode bits when certain opcodes are used in conjunction with certain SPI memory devices. Mode bits are sent using the same number of pins which were used to transmit the address. Once sent, output pins will be held in their final resultant state until the conclusion of all dummy byte periods, unless three-stating the outputs is specified first by the <code>SPI_MMRDH.TRIDMY</code> bits.
14:12 (R/W)	DMYSIZE	Bytes of Dummy/Mode. The <code>SPI_MMRDH.DMYSIZE</code> bit field specifies the number of bytes separating address transmission and read data return. Dummy bytes elapse assuming dummy bits are transmitted using the same number of pins which were used to transmit address.
		0 0 Bytes
		1 1 Bytes
		2 2 Bytes
		3 3 Bytes
		4 4 Bytes
		5 5 Bytes
		6 6 Bytes
		7 7 Bytes

Table 15-25: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	ADRPINS	Pins Used for Address. The <code>SPI_MMRDH.ADRPINS</code> bit specifies the number of pins to be used for address transmission. This bit must be set consistent with expectations established by read opcode. Hardware does not interpret the <code>SPI_MMRDH.OPCODE</code> , but rather relies on this bit to specify behavior.
		0 Use only one pin: MOSI (overrides <code>SPI_CTL.MIOM</code> bits)
		1 Use pins specified by <code>SPI_CTL.MIOM</code> bits
10:8 (R/W)	ADRSIZE	Bytes of Read Address. The <code>SPI_MMRDH.ADRSIZE</code> bit field defines the number of bytes used to specify the read address. The read address is sent immediately following the transmission of opcode. Unlike opcode bits, address bits may be sent using either one or multiple pins. The number of pins is selected using the <code>SPI_MMRDH.ADRPINS</code> bit. The address sent to a connected SPI memory device is an echo of the read address received by the SPI peripheral slave port. The least significant bytes of address are sent when the entire address is not sent.
		0 1 Byte
		1 1 Byte
		2 2 Bytes
		3 3 Bytes
		4 4 Bytes
7:0 (R/W)	OPCODE	Read Opcode. The <code>SPI_MMRDH.OPCODE</code> bit field specifies the initial bits transmitted in response to a read request of SPI memory. Although any opcode may be sent, values 0x03, 0x0B, 0x3B, 0x6B, 0xBB, and 0xEB are likely to be the most commonly used. <code>SPI_MMRDH.OPCODE</code> is sent by the SPI without interpretation; the states of these bits have no effect beyond specifying what is initially shifted across the SPI interface.

SPI Memory Top Address

The `SPI_MMTOP` register specifies the top populated address of a connected SPI memory device.

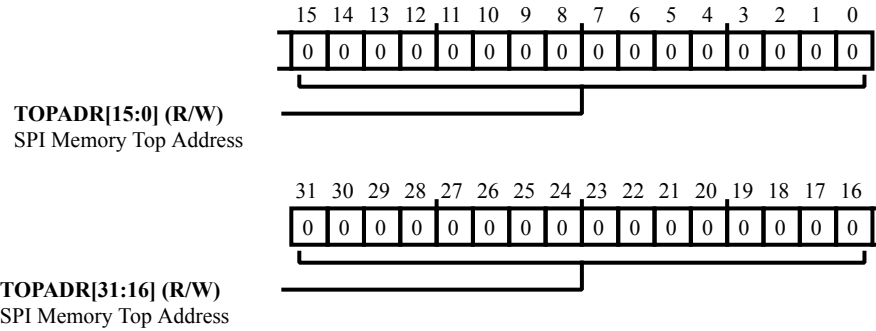


Figure 15-29: SPI_MMTOP Register Diagram

Table 15-26: SPI_MMTOP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TOPADR	SPI Memory Top Address. The <code>SPI_MMTOP.TOPADR</code> bit field specifies the top populated address of a connected SPI memory device. Attempts to access SPI memory are not blocked if this address is exceeded and an error is generated as part of the read response.

Receive FIFO Data Register

The `SPI_RFIFO` register has an interface to the receive shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit `SPI_RFIFO` register, but the size (number of word locations) of the receive FIFO is actually flexible with transfer word size. The size of the receive FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall receive transfers based on FIFO status. When the receive FIFO is full, the SPI master stops initiating new transfers on the SPI if `SPI_RXCTL.RTI` is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data reception continues after `SPI_RFIFO` is full, the data in the receive FIFO is invalid. The SPI indicates this condition with receive overrun (`SPI_STAT.ROR`) error. This condition is possible when `SPI_RXCTL.RTI = 0` and `SPI_RXCTL.REN = 1` for a master, or for a slave that does not exercise flow control.

Note that the receive FIFO is reset (cleared) when the SPI is disabled after being enabled.

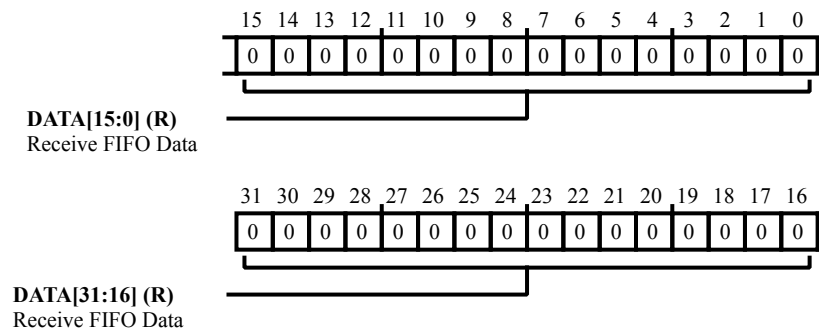


Figure 15-30: `SPI_RFIFO` Register Diagram

Table 15-27: `SPI_RFIFO` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Receive FIFO Data. The <code>SPI_RFIFO.DATA</code> bit field contains the FIFO receive data.

Received Word Count Register

The `SPI_RWC` register holds a count of the number of words remaining to be received by the SPI. To start the decrement of the word count in `SPI_RWC`, enable the receive word counter (`SPI_RXCTL.RWCEN = 1`). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the receive finish interrupt (`SPI_ILAT.RF`). In DMA mode, the SPI uses the `SPI_RWC` register to ensure that the number of frames received during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the `SPI_RWC` registers should match the word count in the DMA configuration. The `SPI_RWC` register maintains the number of frames to be received in a transfer. The `SPI_RWC` should only be changed when the counter is disabled.

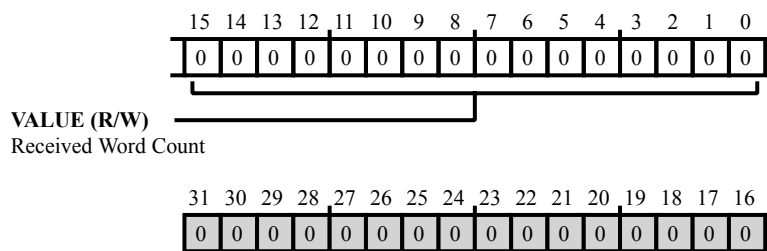


Figure 15-31: SPI_RWC Register Diagram

Table 15-28: SPI_RWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count. The <code>SPI_RWC.VALUE</code> bits hold the receive transfer word count.

Received Word Count Reload Register

The `SPI_RWCR` register holds the receive word count value that the SPI loads into the `SPI_RWC` register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The `SPI_RWCR` register should only be changed when the counter is disabled.

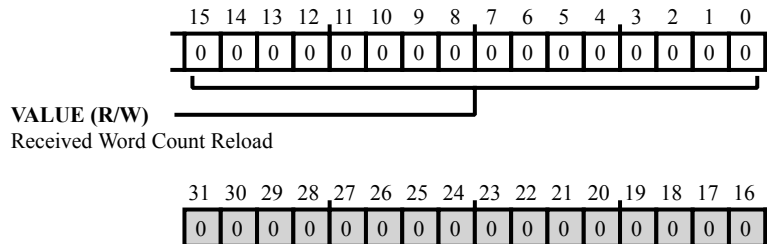


Figure 15-32: SPI_RWCR Register Diagram

Table 15-29: SPI_RWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count Reload. The <code>SPI_RWCR.VALUE</code> bits hold the receive transfer word count reload value.

Receive Control Register

The `SPI_RXCTL` register enables the SPI receive channel, initiates receive transfers, and configures `SPI_RFIFO` buffer watermark settings.

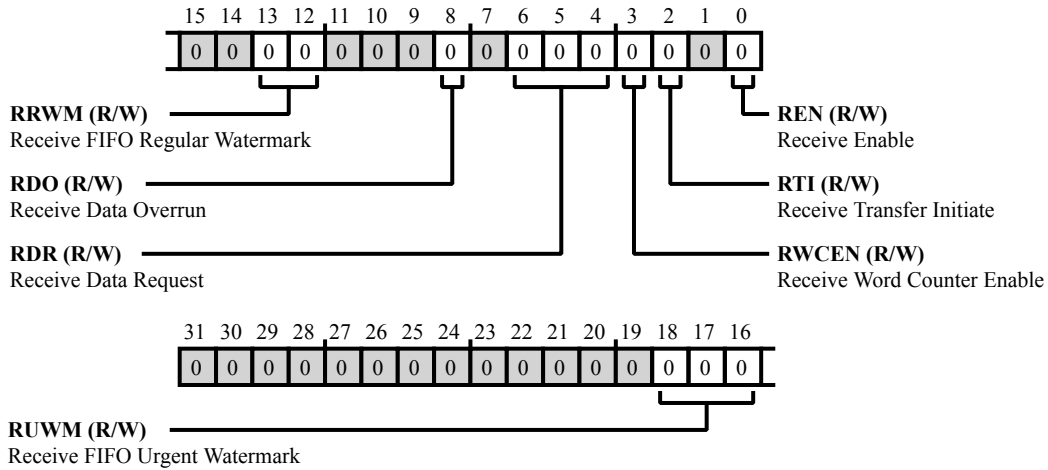


Figure 15-33: SPI_RXCTL Register Diagram

Table 15-30: SPI_RXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:16 (R/W)	RUWM	Receive FIFO Urgent Watermark. The <code>SPI_RXCTL.RUWM</code> bits select the receive FIFO (<code>SPI_RFIFO</code>) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the <code>SPI_ILAT.RUWM</code> interrupt. When an urgent <code>SPI_RFIFO</code> watermark is enabled with <code>SPI_RXCTL.RUWM</code> , the <code>SPI_RXCTL.RRWM</code> selection is used as the deassertion condition for any <code>SPI_ILAT.RUWM</code> interrupts that are latched.
		0 Disabled
		1 25% full RFIFO
		2 50% full RFIFO
		3 75% full RFIFO
		4 Full RFIFO
		5 Reserved
		6 Reserved
7 Reserved		

Table 15-30: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:12 (R/W)	RRWM	Receive FIFO Regular Watermark. The <code>SPI_RXCTL.RRWM</code> bits select the receive FIFO (<code>SPI_RFIFO</code>) watermark level for regular data bus requests. When an urgent <code>SPI_RFIFO</code> watermark is enabled with <code>SPI_RXCTL.RUWM</code> , the <code>SPI_RXCTL.RRWM</code> selection is used as the deassertion condition for any <code>SPI_ILAT.RUWM</code> interrupts that are latched.
		0 Empty RFIFO
		1 RFIFO less than 25% full
		2 RFIFO less than 50% full
		3 RFIFO less than 75% full
8 (R/W)	RDO	Receive Data Overrun. The <code>SPI_RXCTL.RDO</code> bit selects handling for receive data requests when the receive buffer (<code>SPI_RFIFO</code>) is full. If enabled and <code>SPI_RFIFO</code> is full, the SPI overwrites old data in the buffer with incoming data. If disabled and <code>SPI_RFIFO</code> is full, the SPI keeps old data in the buffer and discards incoming data.
		0 Discard incoming data if <code>SPI_RFIFO</code> is full
		1 Overwrite old data if <code>SPI_RFIFO</code> is full
6:4 (R/W)	RDR	Receive Data Request. The <code>SPI_RXCTL.RDR</code> bits select receive FIFO (<code>SPI_RFIFO</code>) watermark conditions that direct the SPI to generate a receive data request.
		0 Disabled
		1 Not empty RFIFO
		2 25% full RFIFO
		3 50% full RFIFO
		4 75% full RFIFO
		5 Full RFIFO
		6 Reserved
		7 Reserved
3 (R/W)	RWCEN	Receive Word Counter Enable. The <code>SPI_RXCTL.RWCEN</code> bit enables the decrement of the <code>SPI_RWC</code> register when the count is not zero and <code>SPI_RXCTL.RTI</code> is enabled. Enabling <code>SPI_RXCTL.RWCEN</code> prevents receive overrun errors from occurring. The <code>SPI_RXCTL.RWCEN</code> bit is valid only when the SPI is a master.
		0 Disable
		1 Enable

Table 15-30: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	RTI	Receive Transfer Initiate. The <code>SPI_RXCTL.RTI</code> bit enables initiation of receive transfers if the receive FIFO (<code>SPI_RFIFO</code>) is not full. The bit also enables this initiation if <code>SPI_RWC</code> is not zero when <code>SPI_RXCTL.RWCEN</code> is enabled. Enabling <code>SPI_RXCTL.RTI</code> prevents receive overrun errors from occurring. The <code>SPI_RXCTL.RTI</code> bit is valid only when the SPI is a master.
		0 Disable
		1 Enable
0 (R/W)	REN	Receive Enable. The <code>SPI_RXCTL.REN</code> bit enables SPI receive channel operation.
		0 Disable
		1 Enable

Slave Select Register

The `SPI_SLVSEL` register enables the `SPI_SEL[n]` pins for output and indicates the state (high or low) of these pins when enabled.

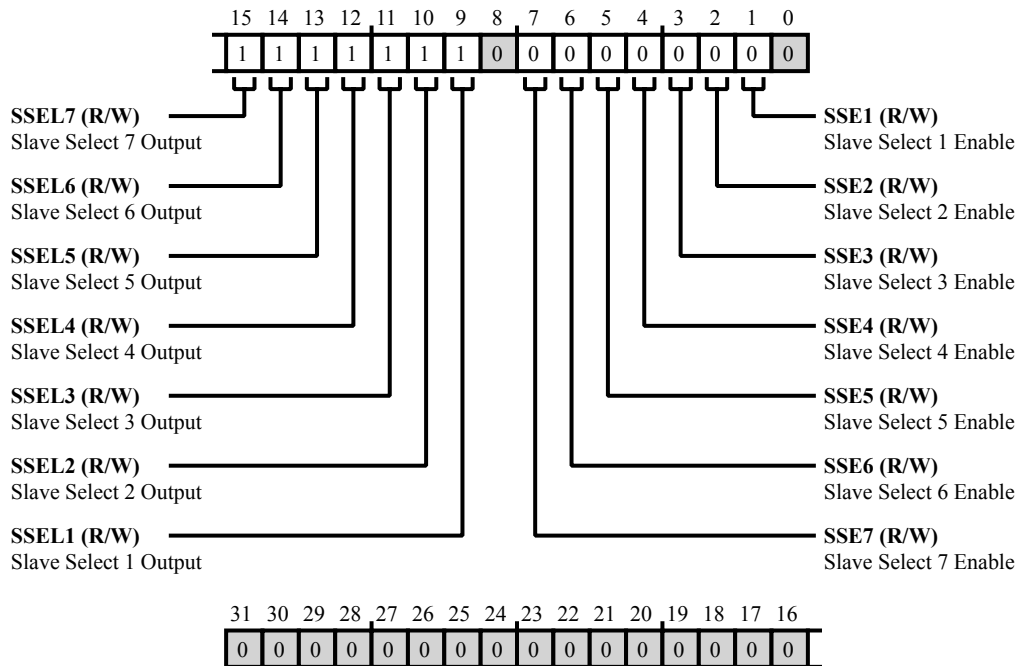


Figure 15-34: SPI_SLVSEL Register Diagram

Table 15-31: SPI_SLVSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	SSEL7	Slave Select 7 Output. The <code>SPI_SLVSEL.SSEL7</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
14 (R/W)	SSEL6	Slave Select 6 Output. The <code>SPI_SLVSEL.SSEL6</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High

Table 15-31: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	SSEL5	Slave Select 5 Output. The <code>SPI_SLVSEL.SSEL5</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
12 (R/W)	SSEL4	Slave Select 4 Output. The <code>SPI_SLVSEL.SSEL4</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
11 (R/W)	SSEL3	Slave Select 3 Output. The <code>SPI_SLVSEL.SSEL3</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
10 (R/W)	SSEL2	Slave Select 2 Output. The <code>SPI_SLVSEL.SSEL2</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
9 (R/W)	SSEL1	Slave Select 1 Output. The <code>SPI_SLVSEL.SSEL1</code> bit state indicates the value driven on the related <code>SPI_SEL[n]</code> pin.
		0 Low
		1 High
7 (R/W)	SSE7	Slave Select 7 Enable. The <code>SPI_SLVSEL.SSE7</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. If disabled, the SPI three-states the related <code>SPI_SEL[n]</code> pin.
		0 Disable
		1 Enable

Table 15-31: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	SSE6	Slave Select 6 Enable. The <code>SPI_SLVSEL.SSE6</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable
5 (R/W)	SSE5	Slave Select 5 Enable. The <code>SPI_SLVSEL.SSE5</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable
4 (R/W)	SSE4	Slave Select 4 Enable. The <code>SPI_SLVSEL.SSE4</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable
3 (R/W)	SSE3	Slave Select 3 Enable. The <code>SPI_SLVSEL.SSE3</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable
2 (R/W)	SSE2	Slave Select 2 Enable. The <code>SPI_SLVSEL.SSE2</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable
1 (R/W)	SSE1	Slave Select 1 Enable. The <code>SPI_SLVSEL.SSE1</code> bit enables the related <code>SPI_SEL[n]</code> pin for output. See the <code>SPI_SLVSEL.SSE7</code> bit description for more information.
		0 Disable
		1 Enable

Status Register

The `SPI_STAT` register indicates SPI status including FIFO status, error conditions, and interrupt conditions. When an interrupt condition from this register is unmasked (enabled) by the corresponding bit in the `SPI_IMSK` register, the interrupt request is latched into the corresponding bit in the `SPI_ILAT` register.

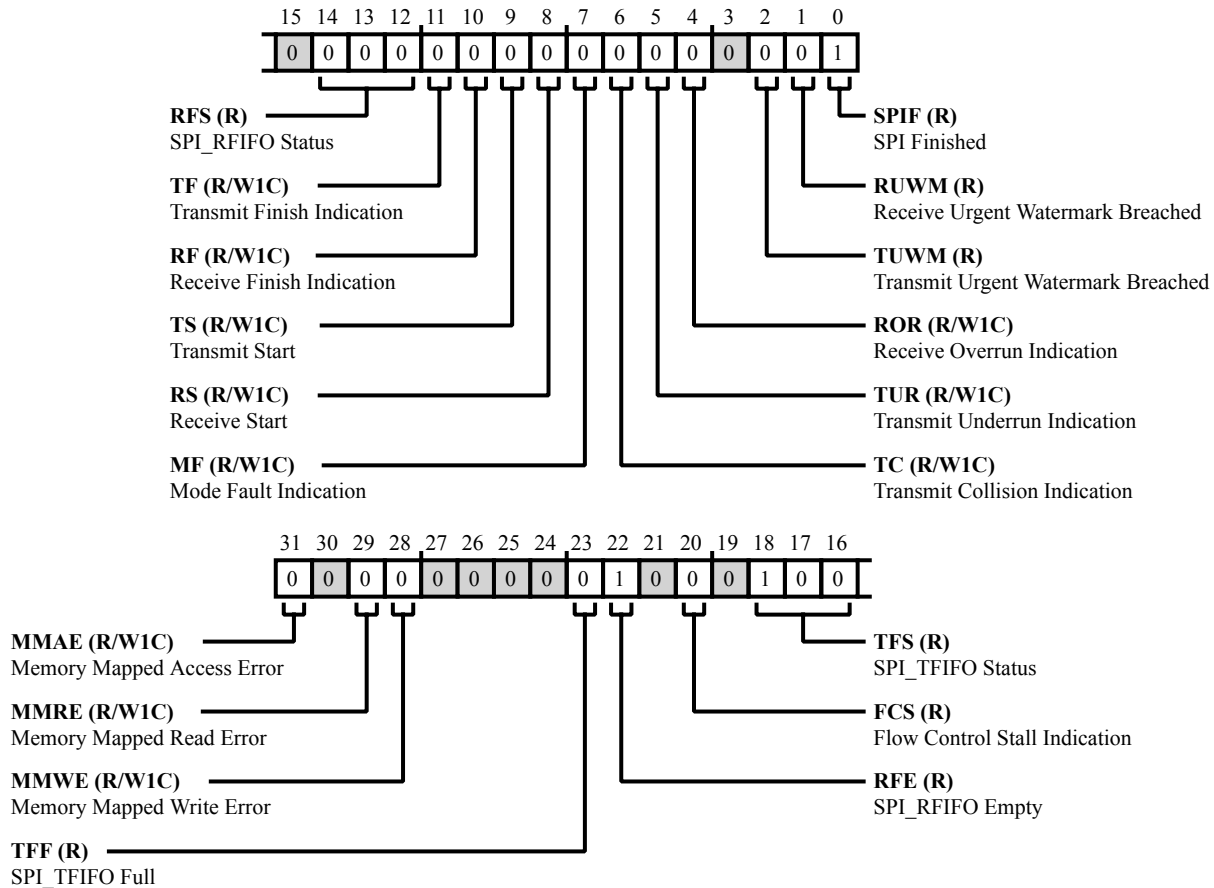


Figure 15-35: `SPI_STAT` Register Diagram

Table 15-32: `SPI_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	MMAE	Memory Mapped Access Error. The <code>SPI_STAT.MMAE</code> bit =1 if an attempt is made to access either the Tx or Rx FIFO while memory-mapped access of SPI memory is enabled (see the <code>SPI_CTL.MMSE</code> bit). The <code>SPI_STAT.MMAE</code> bit =0 when a 1 is written to it. The <code>SPI_STAT.MMAE</code> bit is provided for software notification only. Its state has no further effect.

Table 15-32: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W1C)	MMRE	Memory Mapped Read Error. The <code>SPI_STAT.MMRE</code> bit =1 if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (see the <code>SPI_CTL.MMSE</code> bit). The <code>SPI_STAT.MMRE</code> bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.
28 (R/W1C)	MMWE	Memory Mapped Write Error. The <code>SPI_STAT.MMWE</code> bit =1 if an attempt is made to write address space reserved for memory-mapped SPI memory. The <code>SPI_STAT.MMWE</code> bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.
23 (R/NW)	TFF	SPI_TFIFO Full. The <code>SPI_STAT.TFF</code> bit indicates whether the <code>SPI_TFIFO</code> is full or not full.
		0 Not full Tx FIFO
		1 Full Tx FIFO
22 (R/NW)	RFE	SPI_RFIFO Empty. The <code>SPI_STAT.RFE</code> bit indicates whether the <code>SPI_RFIFO</code> is empty or not empty.
		0 Rx FIFO not empty
		1 Rx FIFO empty
20 (R/NW)	FCS	Flow Control Stall Indication. The <code>SPI_STAT.FCS</code> bit indicates whether a slave has deasserted the <code>SPI_RDY</code> pin to stall the SPI master while the slave is unable to service the SPI masters request. This bit is valid only when the SPI is a master (<code>SPI_CTL.MSTR =1</code>) and flow control is enabled (<code>SPI_CTL.FCEN =1</code>).
		0 No Stall (RDY pin asserted)
		1 Stall (RDY pin deasserted)
18:16 (R/NW)	TFS	SPI_TFIFO Status. The <code>SPI_STAT.TFS</code> bits indicate the status of the <code>SPI_TFIFO</code> . The SPI uses this status when evaluating transmit watermark conditions.
		0 Full TFIFO
		1 25% empty TFIFO
		2 50% empty TFIFO
		3 75% empty TFIFO
		4 Empty TFIFO

Table 15-32: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:12 (R/NW)	RFS	SPI_RFIFO Status. The <code>SPI_STAT.RFS</code> bits indicate the status of the <code>SPI_RFIFO</code> . The SPI uses this status when evaluating receive watermark conditions.
		0 Empty RFIFO
		1 25% full RFIFO
		2 50% full RFIFO
		3 75% full RFIFO
		4 Full RFIFO
11 (R/W1C)	TF	Transmit Finish Indication. The <code>SPI_STAT.TF</code> bit indicates that the SPI has detected the finish of a transmit burst transfer (the <code>SPI_TWC</code> count decrements to zero). This condition can only occur when <code>SPI_TXCTL.TTI</code> and <code>SPI_TXCTL.TWCEN</code> are enabled.
		0 No status
		1 Transmit finish detected
10 (R/W1C)	RF	Receive Finish Indication. The <code>SPI_STAT.RF</code> bit indicates that the SPI has detected the finish of a receive burst transfer (the <code>SPI_RWC</code> count decrements to zero). This condition can only occur when <code>SPI_RXCTL.RTI</code> and <code>SPI_RXCTL.RWCEN</code> are enabled.
		0 No status
		1 Receive finish detected
9 (R/W1C)	TS	Transmit Start. The <code>SPI_STAT.TS</code> bit indicates that the SPI has detected the start of a transmit burst transfer. A transmit bursts starts with the load of <code>SPI_TWC</code> from the <code>SPI_TWCR</code> . This condition can only occur when <code>SPI_TXCTL.TTI</code> and <code>SPI_TXCTL.TWCEN</code> are enabled.
		0 No status
		1 Transmit start detected
8 (R/W1C)	RS	Receive Start. The <code>SPI_STAT.RS</code> bit indicates that the SPI has detected the start of a receive burst transfer. A receive bursts starts with the load of <code>SPI_RWC</code> from the <code>SPI_RWCR</code> . This condition can only occur when <code>SPI_RXCTL.RTI</code> and <code>SPI_RXCTL.RWCEN</code> are enabled.
		0 No status
		1 Receive start detected

Table 15-32: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	MF	Mode Fault Indication. The <code>SPI_STAT.MF</code> bit, when SPI is a master and <code>SPI_CTL.PSSE</code> is enabled, indicates that multiple masters have asserted slave select inputs.
		0 No status
		1 Mode fault occurred
6 (R/W1C)	TC	Transmit Collision Indication. The <code>SPI_STAT.TC</code> bit, when SPI is a slave, indicates that the load of data into the shift register has occurred too close to the first transmitting edge of the SPI clock.
		0 No status
		1 Transmit collision occurred
5 (R/W1C)	TUR	Transmit Underrun Indication. The <code>SPI_STAT.TUR</code> bit, when the transmit FIFO (<code>SPI_TFIFO</code>) is empty, indicates that the last word in the transmit FIFO has been re-sent as transmit data. Alternately, it indicates that zero has been sent as transmit data.
		0 No status
		1 Transmit underrun occurred
4 (R/W1C)	ROR	Receive Overrun Indication. The <code>SPI_STAT.ROR</code> bit, when the receive FIFO (<code>SPI_RFIFO</code>) is full, indicates that a word in the receive FIFO has been overwritten with incoming receive data. Alternately, it indicates that incoming receive data has been discarded.
		0 No status
		1 Receive overrun occurred
2 (R/NW)	TUWM	Transmit Urgent Watermark Breached. The <code>SPI_STAT.TUWM</code> bit indicates that the transmit urgent watermark (<code>SPI_TXCTL.TUWM</code>) has been reached. This condition is cleared when the transmit FIFO fills enough to reach the transmit regular watermark (<code>SPI_TXCTL.TRWM</code>).
		0 Tx regular watermark reached
		1 Tx urgent watermark breached
1 (R/NW)	RUWM	Receive Urgent Watermark Breached. The <code>SPI_STAT.RUWM</code> bit indicates that the receive urgent watermark (<code>SPI_RXCTL.RUWM</code>) has been reached. This condition is cleared when the receive FIFO empties enough to reach the receive regular watermark (<code>SPI_RXCTL.RRWM</code>).
		0 Rx regular watermark reached
		1 Rx urgent watermark breached

Table 15-32: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SPIF	SPI Finished. The <code>SPI_STAT.SPIF</code> bit indicates that a single word transfer is complete.	
		0	No status
		1	Completed single word transfer

Transmit FIFO Data Register

The `SPI_TFIFO` register has an interface to the transmit shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit `SPI_TFIFO` register, but the size (number of word locations) of the transmit FIFO is actually flexible with transfer word size. The size of the transmit FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall transmit transfers based on FIFO status. When the transmit FIFO is empty, the SPI master stops initiating new transfers on the SPI if `SPI_TXCTL.TTI` is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data transmit requests continue after `SPI_TFIFO` is empty, the data sent from the transmit FIFO is invalid, and the SPI indicates this condition with transmit underrun (`SPI_STAT.TUR`). This condition is possible when `SPI_TXCTL.TTI = 0` and `SPI_TXCTL.TEN = 1` for a master, or for a slave that does not exercise flow control.

Note that the transmit FIFO is reset (cleared) when the SPI is disabled after being enabled.

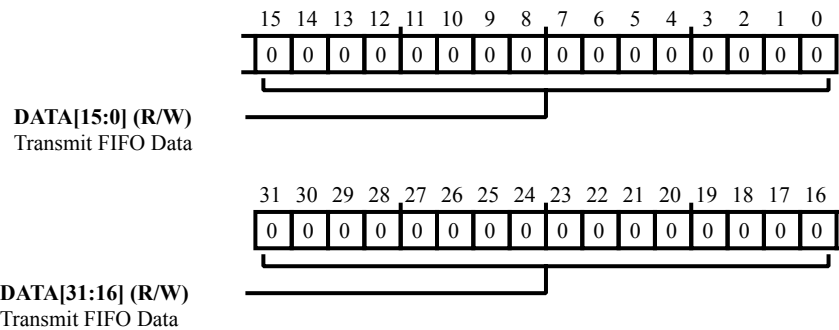


Figure 15-36: SPI_TFIFO Register Diagram

Table 15-33: SPI_TFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit FIFO Data. The <code>SPI_TFIFO.DATA</code> bit field contains the FIFO transmit data.

Transmitted Word Count Register

The `SPI_TWC` register holds a count of the number of words remaining to be transmitted by the SPI. To start the decrement of the word count in `SPI_TWC`, enable the transmit word counter (`SPI_TXCTL.TWCEN = 1`). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the transmit finish interrupt request. In DMA mode, the SPI uses the `SPI_TWC` to ensure that the number of frames transmitted during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the `SPI_TWC` registers should match the word count in the DMA configuration. The `SPI_TWC` maintains the number of frames to be transmitted in a transfer. The `SPI_TWC` should only be changed when the counter is disabled.

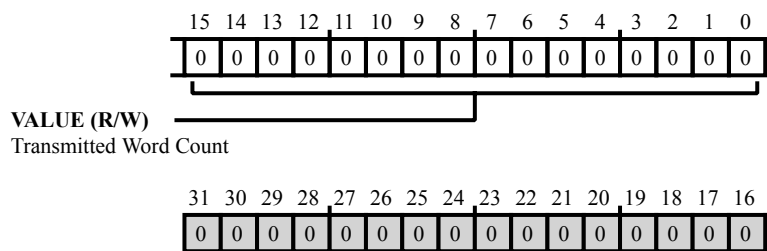


Figure 15-37: SPI_TWC Register Diagram

Table 15-34: SPI_TWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count. The <code>SPI_TWC.VALUE</code> bits hold the transmit transfer word count.

Transmitted Word Count Reload Register

The `SPI_TWCR` register holds the transmit word count value that the SPI loads into the `SPI_TWC` register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The `SPI_TWCR` should only be changed when the counter is disabled.

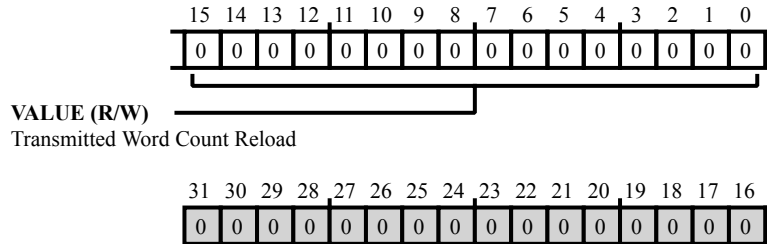


Figure 15-38: SPI_TWCR Register Diagram

Table 15-35: SPI_TWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count Reload. The <code>SPI_TWCR.VALUE</code> bits hold the transmit transfer word count reload value.

Transmit Control Register

The `SPI_TXCTL` register enables the SPI transmit channel, initiates transmit transfers, and configures `SPI_TFIFO` buffer watermark settings.

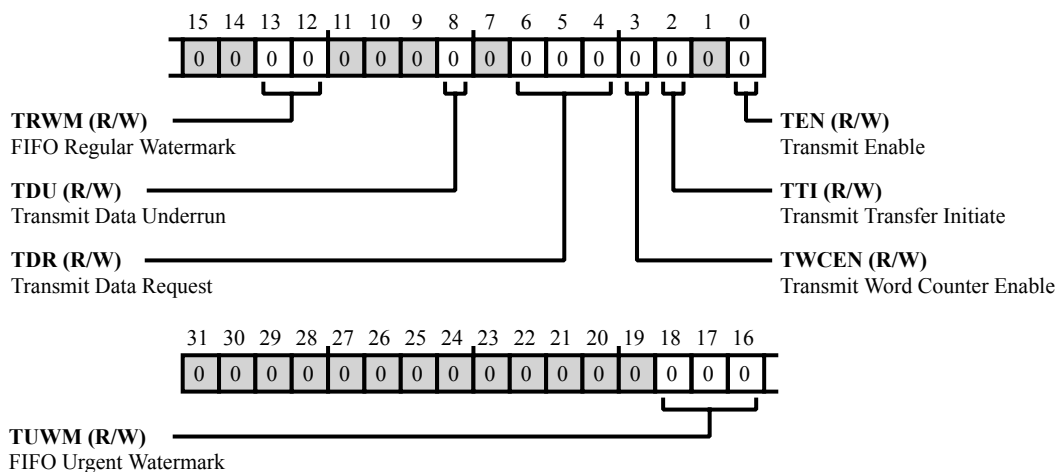


Figure 15-39: SPI_TXCTL Register Diagram

Table 15-36: SPI_TXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:16 (R/W)	TUWM	FIFO Urgent Watermark. The <code>SPI_TXCTL.TUWM</code> bits select the transmit FIFO (<code>SPI_TFIFO</code>) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the <code>SPI_ILAT.TUWM</code> interrupt request. When an urgent <code>SPI_TFIFO</code> watermark is enabled with <code>SPI_TXCTL.TUWM</code> , the <code>SPI_TXCTL.TRWM</code> selection is used as the deassertion condition for any <code>SPI_ILAT.TUWM</code> interrupt requests that are latched.
		0 Disabled
		1 25% empty TFIFO
		2 50% empty TFIFO
		3 75% empty TFIFO
		4 Empty TFIFO

Table 15-36: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:12 (R/W)	TRWM	FIFO Regular Watermark. The SPI_TXCTL.TRWM bits select the transmit FIFO (SPI_TFIFO) watermark level for regular data bus requests. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the deassertion condition for any SPI_ILAT.TUWM interrupt requests that are latched.
		0 Full TFIFO
		1 TFIFO less than 25% empty
		2 TFIFO less than 50% empty
		3 TFIFO less than 75% empty
8 (R/W)	TDU	Transmit Data Underrun. The SPI_TXCTL.TDU bit selects handling for transmit data requests when the transmit buffer (SPI_TFIFO) is empty. If enabled and SPI_TFIFO is empty, the SPI transmits zero as data. If disabled and SPI_TFIFO is empty, the SPI transmits the last word in the buffer as data.
		0 Send last word when SPI_TFIFO is empty
		1 Send zeros when SPI_TFIFO is empty
6:4 (R/W)	TDR	Transmit Data Request. The SPI_TXCTL.TDR bits select transmit FIFO (SPI_TFIFO) watermark conditions that direct the SPI to generate a transmit status interrupt request.
		0 Disabled
		1 Not full TFIFO
		2 25% empty TFIFO
		3 50% empty TFIFO
		4 75% empty TFIFO
		5 Empty TFIFO
3 (R/W)	TWCEN	Transmit Word Counter Enable. The SPI_TXCTL.TWCEN bit enables the decrement of the transmit word count (SPI_TWC) register when the count is not zero and SPI_TXCTL.TTI is enabled. Enabling SPI_TXCTL.TWCEN prevents transmit underrun errors from occurring. The SPI_TXCTL.TWCEN bit is valid only when the SPI is a master.
		0 Disable
		1 Enable

Table 15-36: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	TTI	Transmit Transfer Initiate. The <code>SPI_TXCTL.TTI</code> bit enables initiation of transmit transfers if the transmit FIFO (<code>SPI_TFIFO</code>) is not empty. The bit also enables this initiation if <code>SPI_TWC</code> is not zero when <code>SPI_TXCTL.TWCEN</code> is enabled. Enabling <code>SPI_TXCTL.TTI</code> prevents transmit underrun errors from occurring. The <code>SPI_TXCTL.TTI</code> bit is valid only when the SPI is a master.
		0 Disable
		1 Enable
0 (R/W)	TEN	Transmit Enable. The <code>SPI_TXCTL.TEN</code> bit enables SPI transmit channel operation.
		0 Disable
		1 Enable

16 Octal Serial Peripheral Interface (OSPI)

The Octal Serial Peripheral Interface (OSPI) controller provides access to serial flash devices. It is designed to access flash devices with its modes enabling efficient communication, with minimum overhead on software. Standard SPI is supported along with high performance dual, quad, and octal SPI modes where data can be transferred on up to 8 data pins. Along with normal mode of operation in single transfer (STR) mode, the OSPI controller supports dual transfer rate (DTR) mode, where data, address, and commands are transferred on both edges of the serial clock. With direct access mode support, any read/write to the OSPI memory space triggers a read/write to flash memory.

Direct read accesses using core and DMA are possible, with the latter minimizing software overhead. The OSPI controller also provides Software Triggered Instruction Generation (STIG) mode, which can be used to erase the data on flash devices and access volatile and nonvolatile configuration registers, legacy flash status register, and other status/protection registers.

The OSPI provides a glueless hardware interface with SPI flash devices. The OSPI peripheral includes programmable baud rates, clock phase, clock polarity, separate dummy cycles for read and write accesses, data sampling control, programmable page size/block size, and so on. With this independent control of how many lines to use for command, address, and data allows to virtually interface any flash device with the processor through OSPI controller.

OSPI Features

The OSPI controller supports the following features:

- Full-duplex, synchronous, serial interface
- Memory mapped direct mode operation for performing flash data transfers
- STIG mode operation to issue flash commands
- Automatic flash status polling support for flash program through direct access mode
- XIP (Execute in Place) support
- Support for single, dual, quad, or octal modes of operation
- Support for DDR mode and DTR protocol (including Octal DDR protocol)
- Up to 16-bit data transfer in each SPI clock cycle
- Programmable baud rate, clock phase, and polarity

- Programmable interrupt generation
- Programmable chip select control timing
- Programmable dummy cycles for read and write operations
- Tune data capture mechanism to improve high speed operation
- Programmable write protected regions to block system writes

OSPI Functional Description

ADSP-2156x OSPI Register List

The OSPI is a Octal SPI controller.

Table 16-1: ADSP-2156x OSPI Register List

Name	Description
OSPI_CTL	Octal SPI Control Register
OSPI_DLY	Device Delay Register
OSPI_DRICTL	Device Read Instruction Control Register
OSPI_DWICTL	Device Write Instruction Control Register
OSPI_DSCTL	Device Size Control Register
OSPI_FCA	Flash Command Address Register
OSPI_FCCTL	Flash Command Control Register
OSPI_FCMCTL	Flash Command Control Memory Register
OSPI_FCRD_LWR	Flash Command Read Data Register (Lower)
OSPI_FCRD_UP	Flash Command Read Data Register (Upper)
OSPI_FCWD_LWR	Flash Command Write Data Register (Lower)
OSPI_FCWD_UP	Flash Command Write Data Register (Upper)
OSPI_IMSK	Interrupt Mask Register
OSPI_ISTAT	Interrupt Status Register
OSPI_WRPROT_LWR	Lower Write Protection Register
OSPI_MBCTL	Mode Bit Control Register
OSPI_MODID	Module ID Register
OSPI_POLLEXP	Polling Expiration Register
OSPI_OE_LWR	Opcode Extension Register (Lower)
OSPI_OE_UP	Opcode Extension Register (Upper)
OSPI_POLSTAT	Polling Flash Status Register

Table 16-1: ADSP-2156x OSPI Register List (Continued)

Name	Description
OSPI_RDC	Read Data Capture Register
OSPI_REMAPADDR	Remap Address Register
OSPI_WRPROT_UP	Upper Write Protection Register
OSPI_WRPROT_CTL	Write Protection Control Register
OSPI_WCCTL	Write Completion Control Register

ADSP-2156x OSPI Interrupt List

Table 16-2: ADSP-2156x OSPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
73	OSPI0_INT	OSPI0 Error	Level	

OSPI Block Diagram

The OSPI module is comprised of:

- Low level SPI protocol controller
- Internal transmit/receive FIFOs
- DAC
- STIG controller
- Register interface

The figure shows the OSPI controller block diagram.

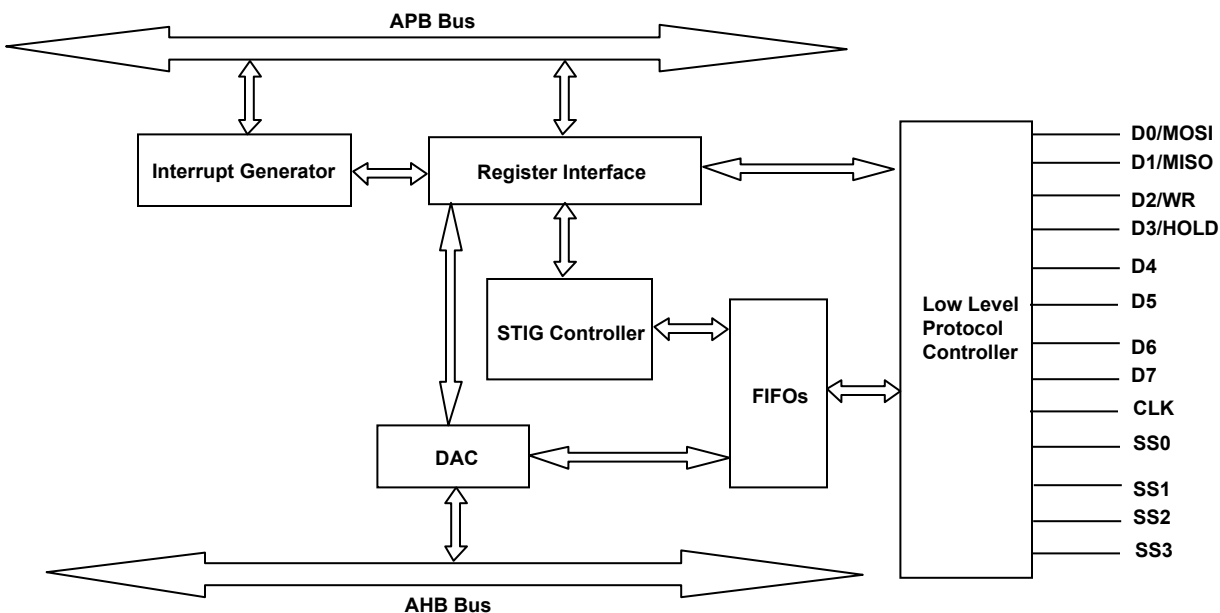


Figure 16-1: OSPI Controller Block Diagram

Architectural Concepts

There are two clock sources for the OSPI controller and one generated clock for clocking external flash device.

The SYSCLK is the main system clock used to transfer data over the AHB bus between the master (core/DMA) and the OSPI controller. The SCLK0_0 clock is used to access the OSPI controller registers to perform basic configuration of the controller and interrupt handling. The SYSCLK, which also acts as a reference clock, is used to serialize the data and drive the OSPI interface. The external clock driven on the OSPI_CLK pin, which is synchronous to the reference clock, is derived from the reference and divided using internal baud rate dividers. Using the reference clock, allows the core to decouple the frequency of the SPI flash device from system clocks, thereby providing a flexible clocking solution.

The OSPI controller supports two independent controllers (to support different types of flash accesses):

- Direct Access Controller (DAC)
- Software Triggered Instruction Generator (STIG)

Direct Access Controller (DAC)

Direct access refers to the operation where AHB bus accesses targeted to the OSPI memory mapped space of processor directly triggers a read/write on flash memory. It is memory mapped and used to access and directly execute the code from external flash memory. When DAC is enabled through the `OSPI_CTL.DACEN` field, any incoming AHB access is serviced by the DAC. Direct accesses can be launched by accessing the OSPI memory address space using core or via one of the MDMA channels. In memory mapped mode, communication to a flash device is automated. The flash memory is accessible directly through reads/write of the processor address space. This access allows code to execute directly from flash devices (true XIP operations). The content can be cached for good performance.

When servicing AHB reads, the DAC sends an additional access downstream apart from the one issued by the AHB in single AHB burst. This is invisible from the system interface. It is a predicted read and ensures that the SPI core operates with maximum bandwidth. The AHB burst not defined by AMBA. It is defined as follows:

- The first access of the AHB burst is defined by:
 - An AHB access that is non-sequential (based on address comparison).
 - An AHB access that is non-sequential or sequential when the downstream modules are idle.
- The last access of the AHB burst is defined by an AHB access that is sequential and precedes a new burst.
- Size of the AHB burst is the number of AHB accesses (first access to last access).
- The number of DAC requests per AHB burst is equal to the size of the AHB burst + 1.

When AHB clock is slower than the SPI data rate, the sequential read transfer may be interrupted on the SPI side and continues after the AHB clock accepts the last data. The limiting value gets narrower as the SPI clock divider and AHB access size decrease and the number of data IOs (single, dual, quad, or octal) increase. If the system cannot meet the SPI data rate, use higher SPI clock divider. This decreases the SPI data rate, but prevents OSPI accesses to external flash to break in between and pass the *Opcode-Address-Dummy* sequence each time. This increases the overall performance by keeping the data transfer continuous.

The controller is designed to work at high data rates. This predicted read results in redundant data during mixed AHB accesses on slow system data rate. Mixed accesses implies that direct AHB sequence of single non-sequential access following few sequential accesses. The predicted read cannot be dropped because of SPI transfer interruption. It is sent as a separate SPI transaction which creates redundant data on the interface for one access. To avoid this issue, drop this data by system or introduce wait states on AHB before issuing a non-sequential direct AHB access.

For AHB writes, the DAC triggers a sequence of write commands that mimic the way reads are processed, though the number of DAC write requests are equal to the number of AHB write requests received. The AHB controller ensures that flash bursts do not break the flash page boundary. When a page boundary is detected, only the number of byte accesses up to that boundary are forwarded. A sequential direct write request that crosses a page boundary must be detected as non-sequential, causing the controller to force the flash device to enter a self-timed page program cycle. The controller supports splitting the writes crossing the page boundaries only for word-aligned addresses.

The external AHB master supplying data for writes must guarantee write data for a particular page that can be provided to the controller in a timely fashion to avoid downstream starvation. If there is a large delay in the system issuing a sequential write, flash write cycle may be prematurely initiated, reducing the device life time.

Flash erase operations, which may be required before a page write, are triggered by software using the STIG. Once a page program cycle has been started, the OSPI controller automatically polls for write cycle to complete before allowing any AHB accesses to complete. This is achieved by holding the subsequent AHB direct accesses in wait state.

Software Triggered Instruction Generator (STIG)

DAC is primarily used to transfer data. To access the volatile and nonvolatile configuration registers, the legacy Flash SPI Status register, other status/protection registers and perform erase operations, a separate software controller is required.

The STIG is controlled using the `OSPI_FCCTL` register by setting up the command to issue to the flash device. This is a generic controller used to perform instructions that the flash device supports from the extended SPI protocol. The STIG controller command is sent to the flash device and written to the software in the `OSPI_FCCTL.OPCODE` field. This is different from the `OSPI_DRICL.OPCODERD` and `OSPI_DWICL.OPCODEWR` fields, as these fields do not impact the STIG operation.

Configuring instructions that are not compliant with the flash specifications can cause unpredictable behavior of the controller. Bit 0 is used to trigger the command, bit 1 used by software to poll the status of the command execution. For reads, when the command has been serviced (bit 1 toggles from 1 to 0), up to 8 bytes of read data is placed in the `OSPI_FCRD_LWR` and `OSPI_FCRD_UP` registers. For writes, data must be placed in the `OSPI_FCWD_LWR` and `OSPI_FCWD_UP` registers. The STIG completion request can be also checked by the corresponding interrupt. The interrupt indicates that the controller is ready to accept a new STIG request. The STIG completion request is not equivalent to completion on the SPI side. For example, if STIG is configured to the data command only on transmit, data is taken from the corresponding STIG register fields and put into TX FIFO. As all write bytes are known, another STIG can be queued before completing serialization of the current one.

There are few commands that require more data to read than 8 bytes (for example, READ ID command). Additional STIG memory bank is implemented to accommodate this data, if required. The STIG memory bank is controlled by the `OSPI_FCMCTL` register. If enabled, the number of bytes to read in the STIG is extended to 16 bytes, as defined in the `OSPI_FCMCTL.RDSZ` field. There are few commands (excluding read arrays that are not intended to be handled in STIG mode, but in Direct Access Mode) that return more than 8 bytes to the controller.

If the number of read bytes in the STIG, as defined in the `OSPI_FCMCTL.RDSZ` field, exceeds the memory bank depth, the remaining data overwrites the STIG memory bank locations starting from its first address. When memory bank is enabled, the `OSPI_FCRD_LWR` and `OSPI_FCRD_UP` registers keep the last 8 bytes read from the flash device by the STIG. For example, to get a single byte from the last eight bytes of a long continuous read SPI data chain, there is no need to access the STIG memory bank as data can be obtained from the `OSPI_FCRD_LWR` and `OSPI_FCRD_UP` registers. To access more data, STIG memory bank data request must be triggered. It is controlled by the `OSPI_FCMCTL` register and works analogously to trigger the STIG. Bit 0 (`OSPI_FCMCTL.TRIGREQ`) is used to trigger the command. Bit 1 (`OSPI_FCMCTL.BNKREQ`) used by software to poll the status of the command execution. When bit 1 toggles from 1 to 0, the data byte (`OSPI_FCMCTL.BNKDATA`) from the corresponding address (`OSPI_FCMCTL.BNKADDR`) is valid. The address must be set before triggering the STIG memory bank access. Each consecutive STIG access overwrites the previous one so that the data in the bank fits into byte index fetched by the last STIG access configured to use the memory bank (first incoming byte equals first address of the memory bank, second one equals the second address and so on).

NOTE: In ADSP-21569, the STIG memory bank depth is 16 bytes.

Servicing STIG Request

The OSPI controller determines the number of bytes in the `OSPI_FCCTL` register to be sent to the flash device, when there is a STIG request. The `OSPI_FCCTL.OPCODE` field indicates the instruction to be sent and is always pushed first. If there is an address to send, the address (size is also programmed in the same register) is sent next. The address is stored in the `OSPI_FCA` register. If there are dummy cycles (size of which is also programmed in the `OSPI_FCCTL` register), they are sent next.

For write data (size of which is also programmed in the `OSPI_FCCTL` register), up to 8 bytes can be sent (as stored in the `OSPI_FCWD_LWR` and `OSPI_FCWD_UP` registers). When read data is collected from the flash device, the OSPI controller stores it in the `OSPI_FCRD_LWR` and `OSPI_FCRD_UP` registers.

Arbitration Between DAC and STIG Access

When multiple controllers are simultaneously active, a simple, fixed priority arbitration scheme is used to arbitrate between each interface and to access the external flash.

The fixed priority is defined as follows, highest priority first:

1. Direct Access Write
2. STIG Access
3. Direct Access Read

When one controller is servicing the request, the other controller is back pressured while waiting to be serviced.

Auto Polling for DAC Write Access

For DAC write accesses, which initiate flash program operation, the OSPI controller provides an option to automatically poll the flash status for program completion. When a flash write operation is triggered through DAC, the OSPI controller sends the write data up to page size defined in the `OSPI_DSCTL.PGSZ` field. Once a page boundary is reached, the DAC stops sending data to the flash device as the flash device enters the program mode. Once the device is in program mode, the OSPI controller can automatically send the flash status read commands to check the state of the flash.

The `OSPI_WCCTL` register defines the setting to poll the flash device correctly. By default, it is configured to poll bit 0 of the Device STATUS register (using opcode 0x05), which is common across all devices to indicate Write in Progress (WIP). However, in some devices (such as Micron devices, where size is > 512 MB), it is required that the controller polls a different bit of a different device register. This means the controller must issue a different command during this polling phase. For example, in Micron N25Q devices, it is bit 7 of the FLAG STATUS register (opcode 0x70) instead of bit 0 of the STATUS register that must be polled. For this reason, the `OSPI_WCCTL` register has been provided for software to determine the bit and the opcode to use to poll for write completion, and also the number of successive valid polls that should take place.

To ensure that the bandwidth of the device is not affected by continuous read status SPI transactions, prolong the delay between the successive flash status read command by programming the `OSPI_WCCTL.REPDLY` field. This feature is implemented to free up the SPI bandwidth, if required. However, defining this delay is not always desired

as the ready bit indication can come later what impacts the overall performance. This register must be setup while the controller is idle.

When the OSPI controller starts to service a STIG request, it sets the `OSPI_FCCTL.STAT` bit to indicate a command execution is in progress. When the OSPI controller is in the auto polling state, servicing a STIG request is different. Several devices are inaccessible after a program operation until the device has completed that write. Few of them have a possibility to suspend the programming page. It can be controlled by the `OSPI_POLSTAT.STAT` bit, which indicates active auto polling phase. After requesting a STIG, the OSPI controller issues an appropriate opcode to memory. While servicing a STIG (in auto polling phase), the `OSPI_FCCTL.STAT` bit remains steady and other parts of transfer such as address or dummy bits are disabled (to issue program suspend command, only opcode is needed).

SPI Command Translation

Requests issued by the DAC or STIG are translated into a sequence of byte transfers to send downstream (before serializing the flash device). These sequences depend on the requested transfer, except the following example (1-byte non-sequential read):

INSTRUCTION OPCODE > ADDRESS > MODEBYTE > DUMMY CYCLES > 1-DATA BYTE

For sequential accesses, an extra data byte per read is pushed to the flash device on the back of the above sequence, assuming that it can be done without any gap between each transferred byte.

The actual sequence sent to the flash device depends on the requested transfers (non-sequential or sequential), whether the device is configured in XIP mode or not, and the state of the `OSPI_DRICTL` and `OSPI_DWICTL` registers. For writes, the write enable latch (WEL) within the flash device must be high before a write sequence can be issued. The OSPI controller automatically issues a WEL command before triggering a write command via DAC (user need not perform this operation). To increase flexibility and performance, turn off this feature by setting the `OSPI_DWICTL.WELDIS`. The opcode for WEL is 0x06 and common between the devices. However, user can program this using the `OSPI_OE_LWR.XRDBYT/OSPI_OE_UP.FSTBYTWEL` and optionally `OSPI_OE_LWR.XWRBYT/OSPI_OE_UP.XBYTWEL` (flash device requires 2-byte command code for WEL).

When write requests from the DAC are not received and all outstanding requests have been sent, the flash device automatically starts the page program write cycle. Any incoming request at this time is held in wait state until the cycle is completed. The OSPI controller automatically polls the flash device legacy SPI status register to identify when the write cycle has completed. This is achieved by sending the Read Status Register (RDSR) opcode to the flash device and waiting until the device indicates that the write cycle is completed (the Write in Progress bit is cleared to zero and WEL bit is cleared to zero or device ready bit is set to one). The WREN and RDSR device instructions are only sent by the controller. For other specific instruction that the user determines must be sent to the device (for example, if the device needs to be unprotected before a write command is issued), has to be separately handled by issuing flash commands via the STIG.

Hold and Reset Control

There is an option to trigger hold and reset features on the I/Os of the flash device. The hold functionality is common across the devices and takes an alternative function of the DQ3 pin (applicable when device does not operate in quad SPI or DDR modes). The transfer can be held and resumed by the `OSPI_CTL.HLD` bit. Devices with hold

feature on DQ3 need another dedicated pin for hardware reset. Devices without hold feature have an alternative reset on the DQ3 pin, which makes the additional reset pin redundant. The controller supports both variants. The `OSPI_CTL.RSTCFG` field allows the user to configure the hardware reset solution implemented in the device. After configuration, it is possible to trigger hold or reset features using I/Os (`OSPI_CTL.HLD` or `OSPI_CTL.RST` bits).

NOTE: The OSPI controller does not support dedicated reset pin to control the reset of flash devices. Flash devices with reset functionality on DQ3 can be controlled directly via OSPI controller.

After hold activation, the controller is in waiting state. Any other operations must not be requested before de-asserting the `OSPI_CTL.HLD` bit. The hold feature is useful when any SPI transaction needs to be prolonged to adjust it into specific point in time. To check if the hold request is suspended, the `OSPI_CTL.IDLE` bit can be polled for. If SPI is not in the idle state, the transfer is suspended. Software must reset the `OSPI_CTL.HLD` bit before any new SPI transaction. In case hold request is set before starting any transfer, it will be in hold state after it starts.

Hardware reset must be activated when CS is high (no valid transaction is present on SPI bus). It can be checked by polling the `OSPI_CTL.IDLE` bit. If the controller is in the idle state and no other transfer requests are queued, hardware reset can be triggered. Reset feature is useful when any write, program, or erase operation needs to be cancelled. No transfer request is permitted before driving the reset back to inactive. Triggering hold or reset on the DQ3 pin at the time the device is configured to work in Quad SPI or DDR mode, overwrites transfer data on the DQ3 with 0. This behaviour is considered as a software error. Ensure that the flash device is introduced to suitable SPI Mode (by polling its Configuration Register) before triggering alternative DQ3 function.

Flash Instruction Type Support

To send correct read and write opcodes, software must initialize the `OSPI_DRICTL` and `OSPI_DWICTL` registers. These registers include fields to setup the required instruction opcodes that is intended to be used to access the flash (default is basic read and basic page program) as well as the instruction type, edge mode (DDR or SDR) and whether the instruction uses single, dual, quad, or octal pins for address and data transfers.

To ensure that the controller can operate from a reset state, the registers are reset to an opcode compatible with single I/O standard SPI devices. Though it is applicable for both reads and writes, the `OSPI_DRICTL.INSTRTYP` field appears only once (not included in the `OSPI_DWICTL` register). If software sets this to any value other than 0, the `OSPI_DRICTL.ADDRTRNSFR/OSPI_DWICTL.ADDRTRNSFR/OSPI_DRICTL.DATATRNSFR/OSPI_DWICTL.DATATRNSFR` bits become don't care. It is made available to allow software to support less common flash instructions, where opcode, address, and data are sent on 2 or 4 data lines (opcode from several instructions are sent serially to the flash device, even for dual/quad instructions).

Dual Data Rate (DDR) Operations

The OSPI controller supports DDR mode of operation, in which some or all transfer phases happen in DDR mode, where the I/O lines are driven/sampled on both rising and falling edges of the clock.

In DDR mode, the OSPI controller supports the following modes:

- Dual Transfer Rate (DTR) mode, where command is driven in SDR mode and address/dummy cycles/data are driven in DDR mode.

- DTR protocol mode, where everything including command is driven in DDR mode.

For DTR mode, there are specific commands which are recognized by the flash as DTR commands. These are DTR Read commands which can be handled in STR Protocol (but in DTR Mode). This is achieved by setting the `OSPI_CTL.DTREN` bit to 0 and `OSPI_DRICTL.DDREN` bit to 1.

For DTR protocol mode, all normal commands can be sent in DDR fashion, provided the flash is already configured in a mode to accept the commands in DDR mode. Support for DTR protocol provides significant enhancement. This is enabled in the OSPI controller by setting the `OSPI_DRICTL.DDREN` bit to 1. It allows the device to work in DTR mode for each possible command (not only DTR read ones). It can also handle opcode phase in DTR mode, which improves the overall performance.

There are several devices (for example, Micron N25Q512A) that can handle read operations in DTR mode. They can issue and capture data on both rising and falling edges while working with a dedicated command type. This enables the controller to maintain same throughput twice the lower frequency of `SPI_CLK` for STR mode.

In MT25Q family of Micron devices, DTR protocol is implemented. It enables device to handle all commands in DTR mode. DTR Read commands detect DTR mode based on dedicated opcode. Therefore, opcode has to be sent as STR. When DTR protocol is enabled, the device does not need opcode to detect edge mode, as it can recognize based on the volatile or nonvolatile bit in the flash configuration register.

The table illustrates how software must configure the controller for selected specific read and write instruction supported by the different devices.

Table 16-3: Read Configuration Examples

Command	Opcode (Number of Lanes/ Edge Mode)	Address/ Dummy/ Mode (Number of Lanes/ Edge Mode)	Data (Number of Lanes/ Edge mode)	Instruction Type <code>OSPI_DRICTL.INSTRYP</code>	Address Transfer Type <code>OSPI_DRICTL.ADDRTRNSFR</code>	Data Transfer Type <code>OSPI_DRICTL.DATATRNSFR</code>	DDR Bit Enable <code>OSPI_DRICTL.DDREN</code>
READ	1/SDR	1/SDR	1/SDR	0	0	0	0
FAST_READ	1/SDR	1/SDR	1/SDR	0	0	0	0
DTR FAST_READ	1/SDR	1/DDR	1/DDR	0	0	0	1
DOR (Dual output Fast Read)	1/SDR	1/SDR	2/SDR	0	0	1	0
DTR DOR (Dual output Fast Read)	1/SDR	1/DDR	2/DDR	0	0	1	1
DIOR (Dual I/O Fast Read)	1/SDR	2/SDR	2/SDR	0	1	1	0

Table 16-3: Read Configuration Examples (Continued)

Command	Opcode (Number of Lanes/ Edge Mode)	Address/ Dummy/ Mode (Number of Lanes/ Edge Mode)	Data (Number of Lanes/ Edge mode)	Instruction Type OSPI_ DRICTL. INSTRYP	Address Transfer Type OSPI_ DRICTL. ADDRTRNSFR	Data Transfer Type OSPI_ DRICTL. DATATRNSFR	DDR Bit Enable OSPI_ DRICTL. DDREN
DTR DIOR (Dual I/O Fast Read)	1/SDR	2/DDR	2/DDR	0	1	1	1
QOR (Quad output Fast Read)	1/SDR	1/SDR	4/SDR	0	0	2	0
DTR QOR (Quad output Fast Read)	1/SDR	1/DDR	4/DDR	0	0	2	1
QIOR (Quad I/O Fast Read)	1/SDR	4/SDR	4/SDR	0	2	2	0
DTR QIOR (Quad I/O Fast Read)	1/SDR	4/DDR	4/DDR	0	2	2	1
OOR (Octal output Fast Read)	1/SDR	1/SDR	8/SDR	0	0	3	0
DTR OOR (Octal output Fast Read)	1/SDR	1/DDR	8/DDR	0	0	3	1
OIOR (Octal I/O Fast Read)	1/SDR	8/SDR	8/SDR	0	3	3	0
DTR OIOR (Octal I/O Fast Read)	1/SDR	8/DDR	8/DDR	0	3	3	1
DCFR (Dual Command Fast Read)	2/SDR	2/SDR	2/SDR	1	don't care	don't care	0
DTR DCFR (Dual Command Fast Read)	2/SDR	2/DDR	2/DDR	1	don't care	don't care	1
QCFR (Quad Command Fast Read)	4/SDR	4/SDR	4/SDR	2	don't care	don't care	0

Table 16-3: Read Configuration Examples (Continued)

Command	Opcode (Number of Lanes/ Edge Mode)	Address/ Dummy/ Mode (Number of Lanes/ Edge Mode)	Data (Number of Lanes/ Edge mode)	Instruction Type OSPI_DRICTL. INSTRYP	Address Transfer Type OSPI_DRICTL. ADDRTRNSFR	Data Transfer Type OSPI_DRICTL. DATATRNSFR	DDR Bit Enable OSPI_DRICTL. DDREN
DTR QCFR (Quad Command Fast Read)	4/SDR	4/DDR	4/DDR	2	don't care	don't care	1
OCFR (Octal Command Fast Read)	8/SDR	8/SDR	8/SDR	3	don't care	don't care	0
DTR OCFR (Octal Command Fast Read)	8/SDR	8/DDR	8/DDR	3	don't care	don't care	1

NOTE: This data is applicable for 3-byte and 4-byte address variants of the commands.

In DTR protocol, all transfer phases (including opcode) take DDR edge mode independent of the command under execution. DTR protocol is enabled by the OSPI_CTL.DTREN bit. It has higher priority than the OSPI_DRICTL.DDREN bit. Therefore, if the OSPI_CTL.DTREN bit is set, irrespective of value programmed in the OSPI_DRICTL.DDREN bit, all command, address, dummy cycles, and data are transferred in DDR mode.

Table 16-4: Write Configuration Examples

Command	Opcode (Number of Lanes)	Address/ Dummy/ Mode (Number of Lanes)	Data (Number of Lanes)	Instruction OSPI_DRICTL. INSTRYP	Address Transfer Type OSPI_DWICTL. ADDRTRNSFR	Data Transfer Type OSPI_DWICTL. DATATRNSFR
PP (Page Program)	1	1	1	0	0	0
DIFP (Dual Input Fast Program)	1	1	2	0	0	1
DIEFP (Dual Input Extended Fast Program)	1	2	2	0	1	1
QIFP (Quad Input Fast Program)	1	1	4	0	0	2

Table 16-4: Write Configuration Examples (Continued)

Command	Opcode (Number of Lanes)	Address/Dummy/Mode (Number of Lanes)	Data (Number of Lanes)	Instruction OSPI_DRICTL .INSTRTYP	Address Transfer Type OSPI_DWICTL .ADDRTRNSFR	Data Transfer Type OSPI_DWICTL .DATATRNSFR
QIEFP (Quad Input Extended Fast Program)	1	4	4	0	2	2
OIFP (Octal Input Fast Program)	1	1	8	0	0	3
OIEFP (Octal Input Extended Fast Program)	1	8	8	0	3	3
DCPP (Dual Command Page Program)	2	2	2	1	don't care	don't care
QCPP (Quad Command Page Program)	4	4	4	2	don't care	don't care
OCPP (Octal Command Page Program)	8	8	8	3	don't care	don't care

NOTE: This data is applicable for both 3-byte or 4-byte address variants of the commands.

In DTR protocol, all transfer phases (including opcode) take DDR edge mode independent of the command under execution.

Data Sampling

The OSPI provides flexibility to configure the sampling point of read data with the SPI clock to meet setup and hold timings of the flash devices when operating at higher speed. The OSPI_RDC register provides an option to control the sampling point. In OSPI controller, data received on the OSPI pins is sampled by internal reference clock (SYSCLK). In STR mode, by default (when the OSPI_RDC.DLYRD field is set to 0), the incoming data is on the first falling edge of the SYSCLK in second half of SPI clock. In DTR mode, it is on the first falling edge of the SYSCLK in second quarter of SPI clock.

The OSPI_RDC.SMPLEDG bit selects the edge of the SYSCLK, in which data from flash memory are sampled. This increases the sampling resolution by two times. The OSPI_RDC.DLYRD field controls the additional number of SYSCLK cycles that must be applied to the internal read data capture circuit. Large clock-to-out delay of the flash memory, trace delays, and other device delays may impose an upper limit on the flash clock frequency which is less for the flash memory to operate. To compensate this, software must set this register to a value that guarantees robust data captures.

For example, if baud divider (`OSPI_CTL.BAUD` field) is programmed to 7, it results in divider of 16. Therefore, `SPI_CLK: SYSCLK = 16:1` (every 1 `SPI_CLK` cycle has 16 `SYSCLK` cycles). Assuming that the `OSPI_RDC.DLYRD` bits are programmed to X ,

In STR mode of operation, received data is captured on:

- $(16/2) + X^{\text{th}}$ falling edge of `SYSCLK` if `OSPI_RDC.SMPLEDG = 0`
- $(16/2) + X^{\text{th}}$ rising edge of `SYSCLK` if `OSPI_RDC.SMPLEDG = 1`

In DTR mode of operation, received data is captured on:

- $(16/4) + X^{\text{th}}$ falling edge of `SYSCLK` if `OSPI_RDC.SMPLEDG = 0`
- $(16/4) + X^{\text{th}}$ rising edge of `SYSCLK` if `OSPI_RDC.SMPLEDG = 1`

To improve the hold timing during transfers in DTR mode, the `OSPI_RDC.DDRDLYRD` delays the transmitted data by programmable number of `SYSCLK` cycles.

NOTE: The OSPI interface for Octal DTR mode may not work at higher frequencies for flash devices that require DQS.

Programming Concepts

Software configures the OSPI controller before communicating with the flash device. The static configuration bits must be setup before the OSPI controller is enabled using the `OSPI_CTL.EN` field. To change the controller configuration, disable it before reconfiguring.

Configuring OSPI after Reset

The OSPI controller comes up with a state that is suitable to perform basic reads and writes using the DAC. Basic read (opcode 0x03) and write (opcode 0x02) instructions are supported by all target devices. The controller also wakes up with a baud rate divider setting of 32. Assuming that the `SYSCLK` (reference clock) operates at 500 MHz after reset, the effective SPI clock is 15.625 MHz. This must be slow enough to meet all timing requirements of all target devices without any further device programming.

If the target device does not use 3 address bytes, the `OSPI_DSCTL` register must be modified to the appropriate size. If software wants writes to the device and the number of bytes per device page is not equal to 256, the `OSPI_DSCTL` register must also be modified. The software must enable the write protect feature prior to enabling the OSPI controller. This blocks any AHB writes from taking effect. The `OSPI_WRPROT_LWR`, `OSPI_WRPROT_UP`, `OSPI_WRPROT_CTL`, and `OSPI_DSCTL.PGSZ` fields must be setup.

After reset, software can read from and write to the flash device. Enabling/disabling the controller and DAC is achieved by writing to the corresponding `OSPI_CTL` register fields. Maintain the default values of the baud rate divisor and default state of the `OSPI_CTL.CPOL/OSPI_CTL.CPHA` bits. A write data value of 0x00780081 is recommended.

Programming Dummy Cycles

When programming the dummy cycles for OSPI reads (DAC and STIG-initiated reads) from the flash device, mode data should be considered as it impacts the actual number of dummy cycles seen on the bus. If mode data is not enabled, then the number of dummy cycles is that same as those set in dummy cycles field, respectively, for STIG or DAC operation.

However, if mode data is enabled, then in addition to the programmed number of dummy cycles, a few extra cycles are needed to transmit out the mode data. The number of cycles added depends on the bus mode of the OSPI. For example, if the address phase of the command is configured to be sent on a single line in STR mode, then the mode bits consume eight cycles. If the address phase of the command is configured to be sent on four lines in DTR mode, then the mode bits consume one cycle. These extra cycles should be considered when configuring the dummy cycles for the controller so the overall dummy clock cycles match the dummy cycles provided in flash data sheet.

Configuring OSPI for Optimal Use

Software must accurately configure the controller to optimally access the flash.

To configure the controller:

1. Wait until pending STIG are completed or poll the `OSPI_CTL.IDLE` field.
2. Disable the `OSPI_CTL.DACEN` field. The OSPI controller can be completely disabled using the `OSPI_CTL.EN` field.
3. Update the `OSPI_DLY` register. This register allows the user to tweak how the chip select is driven after each flash access. This is required as each device may have different timing requirements. As the serial clock frequency is increased, these timing requirements become critical. The numbers programmed in this register are based on the period of reference clock (SYSCLK).

For example, an ATMEL device needs 50 ns minimum time before CS can be re-asserted after it has been de-asserted. By default, the controller provides a minimum of 1 SCLK period. When the device is operating at 50 MHz, the SCLK period is only 20 ns. Therefore, additional 30 ns are required. As the register defines the number of SYSCLK cycles to add, if SYCLK is running at 500 MHz (2 ns period), user must program a value of at least 15 to the `OSPI_DLY.DSRT` bits. This delay can be extended during auto-polling phase. Polling repetition delay can be defined in the `OSPI_WCCTL.REPDLY` field.

4. Update the `OSPI_DSCTL` register. The number of bytes per page is required to perform any write operation on flash. The number of bytes per device block is only required if the write protect feature is used.
5. Setup and enable the write protection registers (`OSPI_WRPROT_LWR`, `OSPI_WRPROT_UP`, and `OSPI_WRPROT_CTL`), if write protection features are required.
6. Enable the required interrupts using the `OSPI_IMSK` register.
7. Update the `OSPI_REMAPADDR` register to remap the DAC addresses to different address in flash.
8. Set the baud rate divisor in the `OSPI_CTL.BAUD` to define the required clock frequency of the target device.

9. Select the appropriate chip select signal to use through the `OSPI_CTL.BAUD` bits.
10. Update the `OSPI_RDC` register. This register delays when read data is captured and helps when read data path from the device to the controller is long and the device clock frequency is high.
11. Enable the OSPI controller and DAC using the `OSPI_CTL` register.

Configuring OSPI for DAC Read Operation

1. Configure the `OSPI_DRICtrl` register:
 - a. Program the desired opcode for Read command (0xBB for Dual SPI read).
 - b. Configure the instruction type, address transfer type, and data transfer type as per the opcode.
 - c. Configure the dummy cycles for the selected command opcode depending on the flash device.
 - d. Set the Mode bit Enable if the mode data needs to be transferred during dummy cycles phase.
 - e. Set the DDR enable bit if the command works in DTR mode.
2. If the operation requires dual opcode, set the `OSPI_CTL.OPCODEEN` bit.
3. Update the second byte for read opcode in the `OSPI_OE_LWR.XRDBYT` field.
4. To operate in DTR protocol mode, set the `OSPI_CTL.DTREN` bit.

If this bit is set, the `OSPI_DRICtrl.DDREN` bit has no effect and all phases of command including the opcode are done in DDR mode.

5. Update the `OSPI_DSCTL.ADDRSZ` field.
6. Update the `OSPI_MBCTL.MODE` field.
7. Enable the DAC mode using the `OSPI_CTL.DACEN` bit, if not enabled.
8. Start the read transfer by accessing the OSPI memory mapped space through core or MDMA accesses. Any access to address 0x60000000 will access the flash address 0x0 if address remapping is 0.
9. If using MDMA for the transfer, wait for the DMA to complete.

NOTE: For accesses through core, software need not check FIFO levels. All the FIFOs are internally maintained by DAC.

Configuring OSPI for DAC Write Operation

1. Configure the `OSPI_DWICtrl` register:
 - a. Program the desired opcode for write command like 0x02 for single SPI page program.
 - b. Configure the address transfer type and data transfer type as per the opcode.
 - c. Configure the dummy cycles for selected command opcode depending on the flash device.

2. Update the `OSPI_DRICtrl.INSTRTYP` field as per the selected opcode.
3. Clear the `OSPI_DWICtrl.WELDIS` bit to manually send the WEL (Write Enable) instruction to flash. Else, the controller automatically issues WEL to flash before sending any write command to flash.
If WEL command for flash under use is not 0x06, update the `OSPI_OE_UP.FSTBYTWEL` field.
4. If the operation requires dual opcode, set the `OSPI_Ctl.OPCODEEN` bit.
5. Update the second byte for read opcode in the `OSPI_OE_LWR.XWRBYT` field.
6. Update the number of address bytes the command expects in the `OSPI_DSCTL.ADDRSZ` field.
7. As the controller supports the automatic polling for write operation complete, update the `OSPI_WCCTL` register accordingly to enable the automatic polling for flash program operation complete.
 - a. Enable the automatic polling by clearing the `OSPI_WCCTL.DIS` bit.
 - b. Update Polling opcode and define polling bit index and polarity as per the flash device.
 - c. Update polling count and polling repetition delay.
 - d. Optionally, update the `OSPI_POLLEXP` register if polling expiration is enabled.
8. Enable the DAC mode using the `OSPI_Ctl.DACEN`, if not enabled.
9. Start the read transfer by accessing the OSPI memory mapped space through core or MDMA accesses. Any access to address 0x60000000 will access the flash address 0x0 if address remapping is 0.
10. Wait for the DMA to complete, if using MDMA for the transfer.

NOTE: For accesses through core, software need not check FIFO levels. All FIFOs are internally maintained by DAC.

Issuing STIG Command

This is a typical method that software uses to access the flash device registers and perform erase operations. It can also be used to access the flash array (though only 8 data bytes can be read or written at a time), as defined in the `OSPI_FCRD_LWR`, `OSPI_FCWD_LWR`, `OSPI_FCRD_UP` and, `OSPI_FCWD_UP` registers.

To issue a STIG command:

1. Set the `OSPI_IMSK.STIGREQ_MSK` bit to enable the STIG command completion interrupt.
2. Update the `OSPI_FCCTL.OPCODE` and `OSPI_FCCTL.DMY` bits, as per the flash command that need to be executed via STIG.
3. Clear the `OSPI_FCCTL.STIGBNKEN` bit.
4. Set the `OSPI_FCCTL.ADDREN` bit, if the command requires the address to be driven.

If set:

- a. Update the `OSPI_FCCTL.ADDRSZ` field according to the number of address bytes expected by the command.
 - b. Update the `OSPI_FCA` register with the desired flash address.
5. If command expects the data to be sent to flash, set the `OSPI_FCCTL.WREN` bit.
- If set:
- a. Update the `OSPI_FCCTL.WRSZ` field according to the number of data bytes to be sent to flash.
 - b. Update the `OSPI_FCWD_LWR` and `OSPI_FCWD_UP` registers with data to be sent to flash.
6. If command expects the data to be received from the flash, set the `OSPI_FCCTL.RDEN` bit.
- If set, update the `OSPI_FCCTL.RDSZ` field according to the number of data bytes to expected from flash.
7. Update the `OSPI_FCCTL.MODEEN` bit, if the mode data are sent during dummy cycles.
- If set, update the `OSPI_MBCTL.MODE` field with desired mode data.
8. Set the `OSPI_FCCTL.EXE` bit to execute the STIG command.
9. Set the `OSPI_ISTAT.STIGREQ` bit (or wait for interrupt if enabled) indicating that the STIG command execution is completed.
10. If command expects the data to be received from the flash, read the data from the `OSPI_FCWD_LWR` and `OSPI_FCWD_UP` registers.

NOTE: Do not set the `OSPI_FCCTL.WREN` and `OSPI_FCCTL.RDEN` bits simultaneously. A given command can either read data or send data to the flash, but not both at the same time.

With normal STIG command, only 8 bytes can be read at a time, but with support of STIG memory bank up to 16 bytes can be read in to STIG memory bank. This is controlled by the `OSPI_FCCTL.STIGBNKEN` bit and `OSPI_FCMCTL` register. Commands issued using this interface have a higher priority than all other read accesses coming from AHB, and interrupts any read commands being requested by the DAC.

To issue a STIG memory bank read command:

1. Set the `OSPI_IMSK.STIGREQ_MSK` bit to enable STIG command completion interrupt.
2. Update the `OSPI_FCCTL.OPCODE` and `OSPI_FCCTL.DMY` bits as per the flash command that need to be executed via STIG.
3. Set the `OSPI_FCCTL.STIGBNKEN` bit.
4. Set the `OSPI_FCCTL.ADDREN` bit, if the command requires the address to be driven.

If set:

- a. Update the `OSPI_FCCTL.ADDRSZ` field according to the number of address bytes expected by the command.

- b. Update the `OSPI_FCA` register with the desired flash address.
5. Update the `OSPI_FCCTL.MODEEN` bit, if the mode data are sent during dummy cycles.
If set, update the `OSPI_MBCTL.MODE` field with the desired mode data.
6. Set the `OSPI_FCCTL.EXE` bit to execute the STIG command.
7. Set the `OSPI_ISTAT.STIGREQ` bit (or wait for interrupt if enabled) indicating that the STIG command execution is completed.

At this stage, 16 bytes of data read from the flash device are available in the STIG memory bank. To access this data, a STIG memory bank read has to be issued. Each read can read one byte from STIG memory bank at a time. To read all data, 16 reads must be issued.

To read a data byte from STIG memory bank:

1. Set the `OSPI_FCMCTL.BNKADDR` field. This can be any value from 0 to 16.
2. Set the `OSPI_FCMCTL.TRIGREQ` bit to trigger the STIG memory bank read.
3. Poll the `OSPI_FCMCTL.BNKREQ` bit to clear.
4. Read the requested data byte from the `OSPI_FCMCTL.BNKDATA`.

Entering XIP mode

The controller supports XIP operations to minimize the latency for back-to-back reads or code execution.

If the flash device comes in XIP on power up, software cannot discover the state of XIP from POR via the Flash Status Register reads. The only operation of a flash device when XIP mode is enabled, is an XIP read. In such cases, software must be aware that if flash enters XIP from POR, the `OSPI_MBCTL.MODE` and `OSPI_CTL.XIPIMM` fields must be set. This makes the controller to enter XIP mode immediately and start communicating with the flash device in XIP mode on next read issued. Therefore, it does not require the read opcode to be transferred. To exit XIP mode, this bit must be set to 0. This takes effect in the attached device only after the next read instruction is executed. Software must ensure that at least one read instruction is requested after resetting this bit before it can be sure XIP mode in the device is exited.

`OSPI_CTL.XIPRD`: If it is not known that the flash device enters XIP from POR, and XIP from POR may be supported by the attached flash device, software can attempt to exit the XIP mode by issuing an XIP exit command using a STIG command. For this, software must be aware of the mode bit requirements of that device, as XIP entry and exit changes per device.

XIP mode is supported in several flash devices. However, flash manufacturers do not have a consistent standard approach. Most of them use signature bits that are sent to the device immediately following the address bytes. Few of them (such as Micron devices) use signature bits and require a flash device configuration register write to enable XIP.

The following section describes how software ensures entry into XIP mode for the flash devices compliant with the OSPI controller.

Micron N25Q, MT25Q, and MT35X Devices

XIP mode must first be enabled by setting the corresponding field of VCR within the flash device. The VCR can be written to using the `OSPI_FCCTL` register to issue a VCR write command.

1. Disable the DAC using the `OSPI_CTL.DACEN` bit to ensure that no new AHB read accesses are sent to the flash device.
2. Use STIG mode to issue a VCR write to flash memory.
3. Set XIP mode bits in the `OSPI_MBCTL.MODE` field to `8'b00000000`.
4. Enable the OSPI controller XIP mode by setting the `OSPI_CTL.XIPRD` bit.
5. Re-enable the DAC.

Micron (Supporting Basic XIP Mode), Winbond, Spansion Devices

1. Disable the DAC using the `OSPI_CTL.DACEN` bit to ensure that no new AHB read accesses are sent to the flash device.
2. Set XIP mode bits in the `OSPI_MBCTL.MODE` to:
 - `8'b10000000`, for Micron devices supporting basic XIP mode
 - `8'b00100000`, for Winbond devices
 - `8'b10100000`, for Spansion devices
3. Enable XIP mode by setting the `OSPI_CTL.XIPRD` bit.
4. Re-enable the DAC.

17 Universal Asynchronous Receiver/Transmitter (UART)

The UART module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

The UART module supports partial modem status and control functionality to allow for hardware flow control.

The UART is a DMA-capable peripheral with separate transmit and receive DMA master channels. The use of DMA requires minimal software intervention as the DMA engine moves the data. The UART can also use a programmed core mode of operation. The core mode requires software management of the data flow using either interrupts or polling.

The UART can use one of the peripheral timers for a hardware-assisted auto-baud detection mechanism. The timers are external to the UART.

NOTE: The UARTs (UART0, UART1 and UART2) operate in the SCLK0 domain.

UART Features

Each UART includes the following features.

- 5–8 data bits
- Programmable extra stop bit and programmable extra half-stop bit
- Even, odd, and sticky parity bit options
- Extra 8-stage receive FIFO with programmable threshold interrupt request
- Flexible transmit and receive interrupt request timing

- Three interrupt outputs for receive, transmit, and status
- Independent DMA operation for receive and transmit
- Programmable automatic request to send (RTS)/clear to send (CTS) hardware flow control
- False start bit detection
- SIR IrDA operation mode
- MDB/ICP v2.0 operation mode
- Internal loopback
- Improved bit rate granularity
- LIN break command/Inter-frame gap transmission support

Table 17-1: UART Specifications

Feature	Availability
<i>Protocol</i>	
Master-Capable	Yes
Slave-Capable	Yes
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	Yes
<i>Access Type</i>	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 (per UART Port)
DMA Descriptor	Yes
Boot Capable	Yes (Slave Mode)
Local Memory	No
Clock Operation	SCLK/16

UART Functional Description

The following sections provide details on the UARTs functionality.

ADSP-2156x UART Register List

The Universal Asynchronous Receiver/Transmitter module (UART) is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word length, stop bit, parity, and interrupt generation options. A set of registers governs UART operations. For more information on UART functionality, see the UART register descriptions.

Table 17-2: ADSP-2156x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

ADSP-2156x UART Interrupt List

Table 17-3: ADSP-2156x UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
93	UART0_TXDMA	UART0 Transmit DMA	Level	20
94	UART0_RXDMA	UART0 Receive DMA	Level	21
95	UART0_STAT	UART0 Status	Level	
96	UART1_TXDMA	UART1 Transmit DMA	Level	34
97	UART1_RXDMA	UART1 Receive DMA	Level	35
98	UART1_STAT	UART1 Status	Level	
99	UART2_TXDMA	UART2 Transmit DMA	Level	37

Table 17-3: ADSP-2156x UART Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
100	UART2_RXDMA	UART2 Receive DMA	Level	38
101	UART2_STAT	UART2 Status	Level	
162	UART0_TXDMA_ERR	UART0 Transmit DMA Error	Level	
163	UART0_RXDMA_ERR	UART0 Receive DMA Error	Level	
164	UART1_TXDMA_ERR	UART1 Transmit DMA Error	Level	
165	UART1_RXDMA_ERR	UART1 Receive DMA Error	Level	
166	UART2_TXDMA_ERR	UART2 Transmit DMA Error	Level	
167	UART2_RXDMA_ERR	UART2 Receive DMA Error	Level	

ADSP-2156x UART Trigger List

Table 17-4: ADSP-2156x UART Trigger List Masters

Trigger ID	Name	Description	Sensitivity
43	UART0_TXDMA	UART0 Transmit DMA	Edge
44	UART0_RXDMA	UART0 Receive DMA	Edge
45	UART1_TXDMA	UART1 Transmit DMA	Edge
46	UART1_RXDMA	UART1 Receive DMA	Edge
47	UART2_TXDMA	UART2 Transmit DMA	Edge
48	UART2_RXDMA	UART2 Receive DMA	Edge

Table 17-5: ADSP-2156x UART Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
82	UART0_TXDMA	UART0 Transmit DMA	Pulse
83	UART0_RXDMA	UART0 Receive DMA	Pulse
84	UART1_TXDMA	UART1 Transmit DMA	Pulse
85	UART1_RXDMA	UART1 Receive DMA	Pulse
86	UART2_TXDMA	UART2 Transmit DMA	Pulse
87	UART2_RXDMA	UART2 Receive DMA	Pulse

ADSP-2156x UART DMA Channel List

Table 17-6: ADSP-2156x UART DMA Channel List

DMA ID	DMA Channel Name	Description
DMA20	UART0_TXDMA	UART0 Transmit DMA
DMA21	UART0_RXDMA	UART0 Receive DMA
DMA34	UART1_TXDMA	UART1 Transmit DMA
DMA35	UART1_RXDMA	UART1 Receive DMA
DMA37	UART2_TXDMA	UART2 Transmit DMA
DMA38	UART2_RXDMA	UART2 Receive DMA

UART Block Diagram

The *UART Block Diagram* figure shows a simplified block diagram of one UART module and how it interconnects to the processor system.

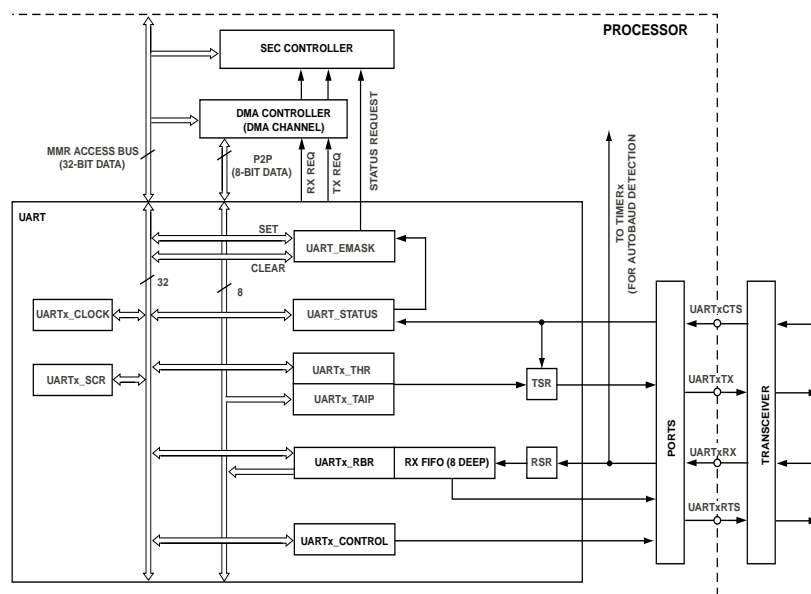


Figure 17-1: UART Block Diagram

UART Architectural Concepts

The following sections provide information about the UART architecture.

Internal Interface

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It operates in either DMA or programmed core modes. The core mode requires software management of the data flow

using either interrupts or polling. The DMA method requires minimal software intervention, as the DMA engine itself moves the data. The `UART_RBR` and `UART_THR` registers also connect to one of the peripheral DMA buses.

All UART registers are 32 bits wide and the registers connect to the peripheral MMR bus. Not all MMRs can be used and unused bits are zero-filled. The UART has three interrupt outputs described as follows.

- The transmit and receive request outputs can function as DMA requests and connect to the DMA controller. Therefore, if the DMA is not enabled, the DMA controller simply forwards the request to the system event controller (SEC).
- The status interrupt output connects directly to the SEC. On many processors, the alternative capture input () of one of the GP timers also senses the `UART_RX` pin. When configured in capture mode, the processor can then use the GP timer to detect the bit rate of the received signal.

External Interface

Each UART features a `UART_RX` (receive) pin and a `UART_TX` (transmit) pin available through the general-purpose ports. These two pins usually connect to an external transceiver device that meets the electrical requirements of full-duplex or half-duplex standards. For example, EIA-232, EIA-422, 4-wire EIA-485 for full-duplex or 2-wire EIA-485, LIN for half-duplex. Additionally, the UART features a pair of clear-to-send, input pins (`UART_CTS`), and request-to-send, output pins (`UART_RTS`) for hardware flow control. UART signals are multiplexed with other functions at the pin level.

Hardware Flow Control

To prevent the UART transmitter from sending data while the receiving counterpart is not ready, the UART features a `UART_RTS/UART_CTS` hardware flow control mechanism. The `UART_RTS` signal is an output that connects to the `UART_CTS` input of the communication partner. If data transfer is bidirectional, the figure shows the *UART Hardware Flow Control* handshake.

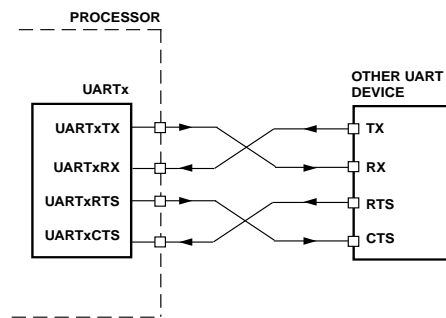


Figure 17-2: UART Hardware Flow Control

In both DMA and core mode, the receiver can deassert the `UART_RTS` signal to indicate that its receive buffer is almost full. Continued data transfers can cause an overrun error. The transmitter pauses when the `UART_CTS` input is in a deasserted state. In this state, the transmitter completes transmission of the data currently held in the transmit shift register (`UART_TSR`) but it does not continue with the data in the transmit hold register (`UART_THR`). If the `UART_CTS` pin is asserted again, the transmitter resumes and loads the content of `UART_THR` register into the `UART_TSR` register.

Bit Rate Generation

The peripheral clock (SCLK) and the 16-bit divisor in the `UART_CLK` register characterize the sample clock. The UART uses the `UART_CTL.EN` bit to enable the clock. By default, every serial bit is oversampled 16 times. The bit clock is 1/16th of the sample clock. If not in IrDA mode, the bit clock can equal the sample clock if the `UART_CLK.EDBO` bit is set, so that the following equation applies:

$$\text{Bit Rate} = \text{SCLK}/16^{(1-\text{EDBO})} \times \text{Divisor}$$

The *UART Bit Rate Examples with 100 MHz SCLK* table provides example divide factors required to support standard baud rates at an SCLK of 100 MHz.

Table 17-7: UART Bit Rate Examples with 100 MHz SCLK

Bit Rate (bits/sec)	EDBO = 0			EDBO = 1		
	DL	Actual	% Error	DL	Actual	% Error
2400	2604	2400.15	0.006	41667	2399.98	0.001
4800	1302	4800.31	0.006	20833	4800.08	0.002
9600	651	9600.61	0.006	10417	9599.69	0.003
19200	326	19171.78	0.147	5208	19201.23	0.006
38400	163	38343.56	0.147	2604	38402.46	0.006
57600	109	57339.45	0.452	1736	57603.69	0.006
115200	54	115740.74	0.469	868	115207.37	0.006
921600	7	892857.14	3.119	109	917431.19	0.452
1500000	4	1562500	4.167	67	1492537.31	0.498
3000000	2	3125000	4.167	33	3030303.03	1.01
6250000	1	6250000	0	16	6250000	0

NOTE: Properly select the SCLK frequencies. Even multiples of bit rate decrease the error percentage.

Setting the bit clock equal to the sample clock (`UART_CLK.EDBO=1`) improves the bit rate granularity and the bit clock matches with the bit rate of the communication partner. The disadvantage is that the power dissipation is higher and sample points are not always accurate. Therefore, use `UART_CLK.EDBO` mode only when bit rate accuracy is not acceptable in the `UART_CLK.EDBO=0` mode.

The `UART_CLK.EDBO=1` mode is not intended to increase the speed of operation beyond the electrical limitations of the UART transfer protocol.

Autobaud Detection

At the chip level, the `UART_RX` pin is typically routed to an alternate capture input () of a general-purpose timer. When working in width capture mode, the processor uses this general-purpose timer to detect the bit rate applied to

the `UART_RX` pin automatically by an external device. It often uses the capture capabilities of the timer to supervise the bit rate at run time. If the UART communicates with any device supplied by a weak clock oscillator that drifts over time, the processor can then readjust its UART bit rate dynamically, as required.

Often, the processor uses autobaud detection for initial bit rate negotiations where it is most likely a slave device waiting for the host to send a predefined autobaud character. This situation is common for UART booting. Do not enable the `UART_CTL.EN` bit while autobaud detection is in-process, to prevent the UART from starting a receive operation with incorrect bit rate matching. Alternatively, set the `UART_CTL.LOOP_EN` bit to disconnect the UART from its `UART_RX` pin.

A software routine can detect the pulse widths of serial stream bit cells. The sample base of the timer is synchronous with the UART operation (all derived from the same `SCLK`). The UART uses pulse widths to calculate the bit rate divider as follows:

$$\text{Divisor} = \text{TIMER_TMR}[n]_WID / (16^{(1-EDBO)} \times \text{Number of captured UART bits})$$

To increase the number of timer counts and the resolution of the captured signal, do not measure just the pulse width of a single bit. Instead, enlarge the pulse of interest over more bits. Traditionally, a NULL character (ASCII 0x00) is used in autobaud detection, as shown in the *Autobaud Detection* figure.

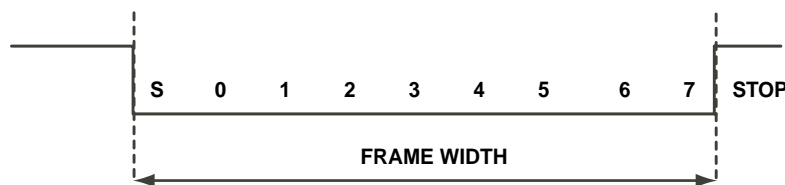


Figure 17-3: Autobaud Detection

Because the example frame encloses 8 data bits and 1 start bit, apply the following formula:

$$\text{Divisor} = \text{TIMER_TMR}[n]_WID / (16^{(1-EDBO)} \times 9)$$

NOTE: For processor-specific mapping of timer alternate capture inputs to the UARTs of the processor, see "Width Capture (WIDCAP) Mode" in the "*General-Purpose Timer (TIMER)*" chapter.

Real receive signals often have asymmetrical falling and rising edges, and the sampling logic level is not exactly in the middle of the signal voltage range. At higher bit rates, such pulse-width-based autobaud detection does not always return adequate results without extra conditioning of the analog signal. Measure signal periods to work around this issue.

For example, predefine the ASCII character "@" (0x40) as the autobaud detection character and measure the period between two subsequent falling edges. As shown in the *Autobaud Detection Character 0x40* figure, measure the period between the falling edge of the start bit and the falling edge after bit 6. Since this period encloses 8 bits, apply the following formula:

$$\text{Divisor} = \text{TIMER_TMR}[n]_PER / (16^{(1-EDBO)} \times 8)$$

or:

- $\text{Divisor} = \text{TIMER_TMR}[n]_PER \gg 7$, if `UART_CLK.EDBO=0`

- Divisor = `TIMER_TMR[n]_PER` >> 3, if `UART_CLK.EDBO=1`

The *Autobaud Detection Character 0x40* figure shows the ASCII “@” (0x40) detection character.

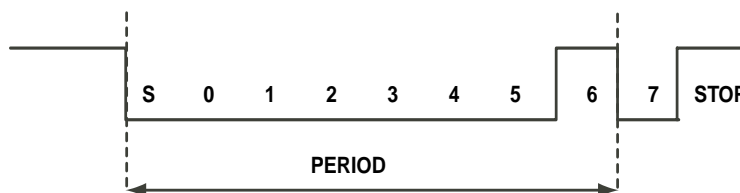


Figure 17-4: Autobaud Detection Character 0x40

UART Debug Features

The UART can automatically calculate and transmit a parity bit. The *UART Parity* table summarizes parity behavior assuming 8-bit data words (`UART_CTL.WLS=b#11`).

Table 17-8: UART Parity

PEN	STP	EPS	Data (hex)	Data (binary, LSB first)	Parity
0	x	x	x	x	None
1	0	0	0x60	0000 0110	1
1	0	0	0x57	1110 1010	0
1	0	1	0x60	0000 0110	0
1	0	1	0x57	1110 1010	1
1	1	0	x	x	1
1	1	1	x	x	0

The two force error bits, `UART_CTL.FPE` and `UART_CTL.FFE`, are intended for test purposes. They are useful for debugging software, especially in loopback mode.

The UART can be set to internal loopback mode (`UART_CTL.LOOP_EN=1`). Loopback mode disconnects the input of the receiver from the receive pin and internally redirects the transmit output to the receiver. The transmit pin remains active and continues to transmit data externally as well. Loopback mode also forces the `UART_RTS` pin to deassert, disconnects the `UART_STAT.CTS` bit from the `UART_CTS` input pin, and connects the internal version of `UART_RTS` to the `UART_STAT.CTS` bit.

Additionally, the `UART_TX` pin can be forced to zero asynchronously using the `UART_CTL.SB` bit.

UART Operating Modes

The following sections describe the main operating modes of the UART.

- [UART Mode](#)
- [IrDA SIR Mode](#)

- [Multi-Drop Bus Mode](#)

UART Mode

The UART mode follows an asynchronous serial communication protocol with these options:

- 1 start bit
- 5–8 data bits
- Address bit (available in MDB mode only)
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits valid only in 5-bit word length)

The `UART_CTL` register controls the format of received and transmitted character frames. Data is always transmitted and received with the least significant bit (LSB) first.

The *Bit Stream on a UART TX Pin Transmitting an “S” Character (0x53)* figure shows a typical physical bit stream measured on a `UART_TX` pin.

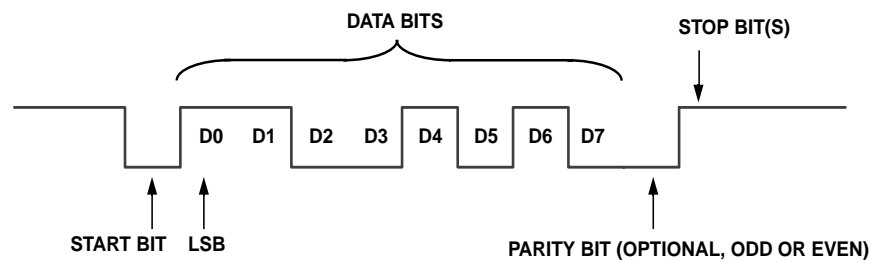


Figure 17-5: Bit Stream on a UART TX Pin Transmitting an “S” Character (0x53)

IrDA SIR Mode

The UART also supports serial data communication by way of infrared signals, according to the recommendations of the Infrared Data Association (IrDA). The physical layer known as IrDA SIR (9.6/115.2 Kbps rate) is based on return-to-zero-inverted (RZI) modulation. The UART does not support pulse position modulation.

Using the 16x data rate clock, RZI modulation is achieved by inverting and modulating the non-return-to-zero (NRZ) code normally transmitted by the UART. On the receive side, the UART uses a 16x clock to determine an IrDA pulse sample window, from which it recovers the RZI modulated NRZ code.

NOTE: The `UART_CLK.EDBO` bit is not valid in IrDA mode. Clear (=0) this bit in this mode.

Multi-Drop Bus Mode

The UART uses a protocol for point-to-point connections as well as in networks where the EIA-485 standard is representative of UART-based bus systems. The EIA-232 standard defines point-to-point connections. In such networks, node addressing is important.

In a multidrop bus (MDB) network, for example, an address bit enhances the UART frame. The address bit is inserted between the data bits and the optional parity bit. To configure the UART for MDB mode, set the mode of operation bits (`UART_CTL.MOD [5:4]`) to 01.

By convention, the address bit is transmitted low for regular data bytes. When set, it marks special address bytes that require the attention of all nodes on the network.

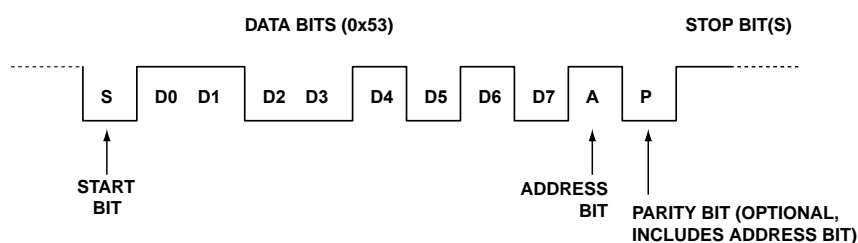


Figure 17-6: UART Frame with Address Bit

All transmit operations are processed through the transmit buffer register (`UART_THR`), so all DMA data transmissions clear the address bit. If data is written to the transmit address or insert pulse register (`UART_TAIP`) instead, the same transmit operation is initiated with the only exception that the address bit is sent high.

The UART uses the `UART_STAT.ADDR` bit of the receiver to signal whether the previously received frame had the address bit set or not. Hardware updates it every time a new frame is received. When the enable address word interrupt bit (`UART_IMSK.EAWI`) is set, the reception of an address byte triggers a special status interrupt request.

The address sticky bit (`UART_STAT.ASTKY`) is the sticky version of the `UART_STAT.ADDR` bit. Hardware sets it whenever the `UART_STAT.ADDR` bit is set. Software can clear the `UART_STAT.ASTKY` bit with a W1C operation.

In MDB mode, only address bytes progress to the receive FIFO by default. Data bytes are gated unless the `UART_STAT.ASTKY` bit is set. The receiver ignores all traffic on the UART bus. This way, the processor can go into low-power mode and interrupt activity does not load the processor every time a frame is transmitted on the UART bus. If, however, an address frame is transmitted, the receiver immediately samples all further traffic. A software routine can analyze the received data, decide whether it was of relevance for the local network node, and W1C the `UART_STAT.ASTKY` bit if it was not.

Software can overrule of the hardware address frame detection by setting the `UART_STAT.ADDR` bit and (indirectly) the `UART_STAT.ASTKY` bit with a W1S operation.

The MDB mode follows an asynchronous serial communication protocol with the following options.

- 1 start bit
- 5–8 data bits
- Address bit
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits are valid only in 5-bit word length)

NOTE: If the address bit and parity bit are both enabled, the parity check and generation includes the address bit in its parity calculation.

UART Data Transfer Modes

The UART can transfer data using both the core and DMA. Receive and transmit paths operate independently except that the bit rate and the frame format are identical for both transfer directions. Transmit and receive channels are both buffered. The `UART_THR` register buffers the transmit shift register (`UART_TSR`) and the `UART_RBR` register buffers the receive shift register (`UART_RSR`).

UART Mode Transmit Operation (Core)

In core mode, the processor core moves data to and from the UART. A write to the `UART_THR` register initiates the transmit operation. If no former operation is pending, the `UART_THR` register passes the data immediately to the `UART_TSR` register. There, it is shifted out at the bit rate characterized by the `UART_CLK` register, with start, stop, and parity bits appended as defined by the `UART_CTL` register.

The `UART_THR` register and the `UART_TSR` register can be modeled as a two-stage transmit buffer. The least significant bit (LSB) always transmits first. This bit is bit 0 of the value written to the `UART_THR` register.

UART Mode LIN Break Command

Some UART-based protocols demand synchronization methods that are not native to standard UART implementations. For example, the Local Interconnect Network (LIN) protocol requires a low-pulse of well-defined transmit length as a prologue to every multi-byte message. Its length must be at least 13 bit-times.

With previous UARTs, there were two options to implement this protocol:

- A null byte is transmitted with a temporarily lowered bit rate, or
- A software counter generates the period and the asynchronous set break (SB) mechanisms pull the transmit pin low

Since both methods have their disadvantages, the newer UART introduces a new inter-frame gap technique.

The feature is not available in MDB or IrDA operating modes. However, in standard UART mode (bits `UART_CTL.MOD[5:4] = 00`), a write to the `UART_TAIP` register initiates the transmission of an inter-frame pulse. If the transmit buffer is not empty, the UART first transmits all bytes in the queue. It only initiates with pulse generation after the last stop bit of the last byte has been shifted out.

The value written into the `UART_TAIP` register defines the nature and the duration of the transmitted pulse. Bits [6:0] control the duration in bit-times and bit [7] controls the value (duration = `UART_TAIP[6:0] / UART_CLK[15:0]`). If `UART_TAIP[7]` is set, and an active high pulse is issued, the number of stop bits is extended. If `UART_TAIP[7]` is cleared, a low pulse is generated. Invert the polarity using the `UART_CTL.FCPOL` bit. Writing a value of 13 into the `UART_TAIP` register generates the break command as required by the LIN protocol.

NOTE: If the `UART_CTL.TPOLC` bit is enabled, an inverted most-significant bit can be transmitted.

NOTE: If another transmission is pending (in the `UART_TSR` register), the `UART_TAIP` initiated pulse is queued until after all pending operations have finished and all stop bits are transmitted.

The transmission of break command/inter-frame gap precedes transmission of the number of stop bits as set in the `UART_CTL.STB` and `UART_CTL.STBH` bit fields.

The UART receiver can detect break commands through the break indicator (`UART_STAT.BI`) flag. This flag reports that an entire UART frame has been received in low state. It does not report whether the duration of the received low pulse was exact or at least 13 bit-times as LIN masters transmit. Typically, the break indicator meets LIN requirements. The processor can use GP timers to determine the pulse width more precisely, if necessary.

Each `UART_RX` pin is also routed to a GP timer through its alternate capture input (TACI). This functionality is not only useful for bit rate detection (*autobaud*) but also helps to measure the pulse widths precisely on the `UART_RX` input. Additionally, the GP timers can issue an interrupt request or a fault condition when the received pulse width is shorter than a bit time or longer than the worst-case break condition. The windowed watchdog width mode of the GP timers controls this functionality.

UART Mode Receive Operation (Core)

The receive operation uses the same data format as the transmit configuration except that one valid stop bit is always sufficient. The `UART_CTL.STB` and `UART_CTL.STBH` bits have no impact on the receiver.

The UART receiver senses the falling edges of the receive input. When it detects an edge, the receiver starts sampling the input according to settings in the `UART_CLK` register. The receiver samples the start bit (majority sampling) close to its midpoint. If sampled low, it assumes a valid start condition. Otherwise, it discards the detected falling edge.

After detection of the start bit, the received word is shifted into the `UART_RSR` register.

After the corresponding stop bit is received, the content of the `UART_RSR` register is transferred to the 8-deep receive FIFO and is accessible by reading the `UART_RBR` register.

The receive FIFOs and the `UART_RBR` register act as a 9-stage receive buffer. If the stop bit of the ninth word is received before software reads the `UART_RBR` register, an overrun error is reported. Overruns protect data in the `UART_RBR` register and the receive FIFO from being overwritten by further data until the software clears the `UART_STAT.OE` bit. However, the data in the `UART_RSR` register is immediately destroyed as soon as the overrun occurs.

The sampling clock is 16 times faster than the bit clock. The receiver oversamples every bit 16 times and makes a majority-decision based on the middle three samples. This functionality improves immunity against noise and hazards on the line. The receiver disregards spurious pulses of less than two times the sampling clock period.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the `UART_CLK.EDBO` bit is set to 1, the receiver samples bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy as required at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

Reception starts when the UART receiver detects a falling edge on the `UART_RX` input pin. The receiver attempts to see a start bit. The data is shifted into the `UART_RSR` register. After the ninth sample of the first, the receiver processes the stop bit and copies the received data to the 8-stage receive FIFO. The `UART_RSR` recovers for further data reception.

The receiver samples data bits close to their midpoint. Because the receiver clock is typically asynchronous to the data rate of the transmitter, the sampling point can drift relative to the center of the data bits. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. The polarity of received data is selectable, using the `UART_CTL.RPOLC` bit.

NOTE: The receiver checks for only a single stop bit. After the third sample of the first stop bit has been received (at time 9/16th of the stop bit duration), the receiver immediately acts (status update). It then prepares for new falling edge detection (start detection).

IrDA Transmit Operation

To generate the IrDA pulse transmitted by the UART, the normal NRZ output of the transmitter is first inverted if the `UART_CTL.TPOLC` bit is configured for active-low operation. In this configuration, a zero is transmitted as a high pulse of 16 UART clock periods and a one is transmitted as a low pulse for 16 UART clock periods. Then, six UART clock periods delay the leading edge of the pulse. Similarly, eight UART clock periods truncate the trailing edge of the pulse. For a 16-cycle UART clock period, this operation results in the final representation of the original zero as a high pulse of only 3/16 clock periods. The *IrDA Transmit Pulse* figure shows how the pulse is centered around the middle of the bit time. The final IrDA pulse is fed to the off-chip infrared driver.

This modulation approach ensures a pulse width output from the UART of three cycles high out of every 16 UART clock cycles. As shown in the *IrDA Transmit Pulse* figure, the error terms associated with the bit rate generator are small and well within the tolerance of most infrared transceiver specifications.

NOTE: In IrDA mode, writes to the `UART_TAIP` register are equivalent to writes to the `UART_THR` register.

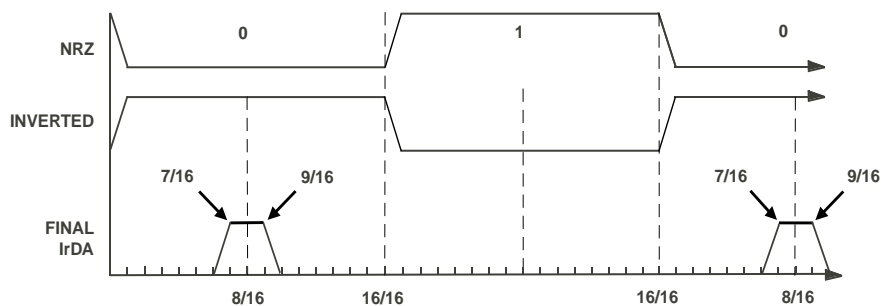


Figure 17-7: IrDA Transmit Pulse

IrDA Receive Operation

The IrDA receiver function is more complex than the transmit function. The receiver must discriminate the IrDA pulse and reject noise. The receiver looks for the IrDA pulse in a narrow window centered around the middle of the expected pulse.

Glitch filtering is accomplished by counting 16 system clocks from the time the receiver detects an initial pulse. If the pulse is absent when the counter expires, the receiver interprets it as a glitch. Otherwise, the receiver interprets it as a zero. This assessment is acceptable because glitches originating from on-chip capacitive cross-coupling typically do not last for more than a fraction of the system clock (SCLK) period. Appropriate shielding avoids sources outside of the chip and not part of the transmitter. The only other source of a glitch is the transmitter itself. The processor relies on the transmitter to perform within specification. If the transmitter violates the specification, unpredictable results can occur. The 4-bit counter adds an extra level of protection at a minimal cost.

NOTE: Because SCLK can change across systems, the longest glitch tolerated is inversely proportional to the SCLK frequency.

A counter that is clocked at the 16x bit-time sample clock determines the receive sampling window. The sampling window is resynchronized with each start bit by centering the sampling window around the start bit.

The polarity of receive data is selectable, using the `UART_CTL.RPOLC` bit. The *IrDA Receiver Pulse Detection* figure provides examples of each polarity type.

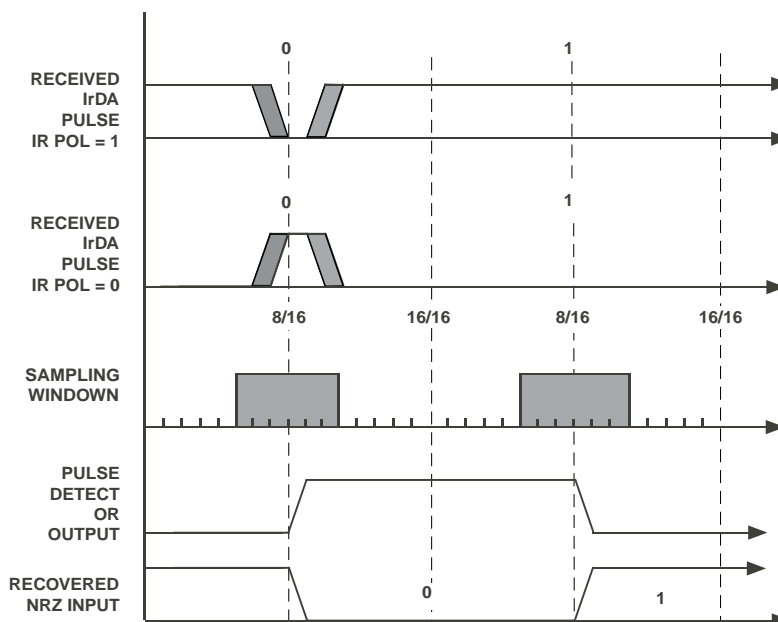


Figure 17-8: IrDA Receiver Pulse Detection

MDB Transmit Operation

In MDB mode, receive and transmit paths operate independently from each other, except for sharing bit rate and frame formats for both transfer directions.

Transmit operation is initiated by writing the `UART_THR` or `UART_TAIP` registers. A write to the `UART_THR` register transmits the written word with the appending address bit set low. A write to the `UART_TAIP` register transmits the written word with the appended address bit set high. The data is moved into the `UART_TSR` register, where it is shifted out at the bit rate programmed by the `UART_CLK` register, with start, stop, address, and parity bits appended, as required.

If DMA is enabled, the DMA engine always writes the data into the `UART_THR` register, and the written word is transmitted with the appending address bit set low.

The polarity of transmit data is selectable, using the `UART_CTL.TPOLC` bit.

MDB Receive Operation

Receive operations use the same data format as the transmit configuration, except that the number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the `UART_RSR` register at the programmed bit.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the `UART_CLK.EDBO` bit is set, the receiver samples the bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy needed at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

After the appropriate number of bits (including address, parity, and stop bits) is received, the `UART_RSR` register is transferred to the receive FIFO and accessible through the `UART_RBR` register.

The polarity of receive data is selectable, using the `UART_CTL.RPOLC` bit.

DMA Mode

In DMA mode, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data; it just has to set up the appropriate transfers either through the descriptor mechanism or through autobuffer mode.

DMA channels provide a 4-deep FIFO, resulting in total buffer capabilities of 6 words at the transmit side and 9 words at the receive side. In DMA mode, the bus activity and arbitration mechanism determine the latency. The processor loading and interrupt priorities do not determine the latency.

To enable UART DMA, first set up the system DMA control registers. Then, enable the `UART_IMSK.ERBFI` or `UART_IMSK.ETBEI` interrupts. This sequence is necessary because these interrupt request lines double as DMA request lines. With DMA enabled, once these requests are received, the DMA control unit generates a direct memory access. If DMA is not enabled, the UART interrupt is passed on to the system interrupt handling unit. The status interrupt for the UART goes directly to the system event controller (SEC), bypassing the DMA unit completely.

For transmit DMA, programs must set the `DMA_CFG.SYNC` bit. With this bit set, interrupt generation is delayed until the entire DMA FIFO is drained to the UART module. The UART transmit DMA interrupt service routine can disable the DMA or to clear the `UART_IMSK.ETBEI` control bit only when the `DMA_CFG.SYNC` bit is set. Otherwise, up to four data bytes can be lost.

When the `UART_IMSK.ETBEI` bit is set, an initial transmit DMA request is issued immediately. The program then clears the `UART_IMSK.ETBEI` bit through the DMA service routine.

In DMA transmit mode, the `UART_IMSK.ETBEI` bit enables the peripheral request to the DMA FIFO. The `DMA_CFG.EN` bit enables the strobe on the memory side. If the DMA count is less than the DMA FIFO depth, which is 4, then the DMA interrupt can be requested before the `UART_IMSK.ETBEI` bit is set. If this behavior is unwanted, set the `DMA_CFG.SYNC` bit.

Regardless of the `DMA_CFG.SYNC` setting, the DMA stream has not left the UART transmitter completely at the time the interrupt request is generated. Transmission can abort in the middle of the stream, causing data loss, when the UART clock was disabled without extra synchronization with the `UART_STAT.TEMT` bit.

The UART provides functionality to avoid resource-consuming polling of the `UART_STAT.TEMT` bit. The `UART_IMSK_SET.EDTPTI` bit enables the `UART_STAT.TEMT` bit to trigger a DMA interrupt. To delay the DMA completion interrupt until the last data word of a STOP DMA has left the UART, keep the `DMA_CFG.INT` bit cleared and set the `UART_IMSK_SET.EDTPTI` bit instead. Then, the normal DMA completion interrupt is suppressed. Later, the `UART_STAT.TEMT` event triggers a DMA interrupt after the last word of the DMA has left the UART transmit buffers. If `DMA_CFG.INT` and `UART_IMSK.EDTPTI` are set, when finishing STOP mode, the DMA requests two interrupts.

The DMA of the UART module supports 8-bit and 16-bit operation, but not 32-bit operation. It does not support sign-extension.

Mixing DMA and Core Modes

Switching from DMA mode to core operation dynamically requires some consideration, especially for transmit operations. By default, the interrupt timing of the DMA is synchronized with the memory side of the DMA FIFOs. Normally, the transmit DMA completion interrupt is generated after the last byte is copied from the memory into the DMA FIFO. The transmit DMA interrupt service routine is not yet permitted to disable the `DMA_CFG.EN` bit. The interrupt is requested when the `DMA_STAT.IRQDONE` bit is set. The `DMA_STAT.RUN` bit, however, remains set until the data has completely left the transmit DMA FIFO.

When planning to switch from a DMA to core mode, set the `DMA_CFG.SYNC` bit in the word of the last descriptor or work unit before handing over control. Then, after the interrupt request occurs, software can write new data into the `UART_THR` register as soon as the `UART_STAT.THRE` bit permits. If the `DMA_CFG.SYNC` bit cannot be set, software can poll the `DMA_STAT.RUN` bit instead. Alternatively, using the `UART_IMSK.EDTPTI` bit can avoid expensive status bit polling.

When switching from core to DMA operation, ensure that the first DMA request is issued properly. If the DMA is enabled while the UART is still transmitting, no precaution is required. If, however, the DMA is enabled after the `UART_STAT.TEMT` bit is high, pulse the `UART_IMSK.ETBEI` bit to initiate DMA transmission.

Setting Up Hardware Flow Control

The following steps show how to set up UART hardware flow control:

1. Configure automatic or manual hardware flow control for the receiver through the `UART_CTL.ARTS` bit, or the transmitter through the `UART_CTL.ACTS` bit.
2. Configure `UART_CTS` and `UART_RTS` polarity through the `UART_CTL.FCPOLE` bit.

On reset, when the UART is not yet enabled and the port multiplexing has not been programmed, the `UART_RTS` pin is not driven. Some applications require a resistor to pull the `UART_RTS` signal to either state during reset.

UART Event Control

Status flags in the `UART_STAT` register are available to signal data reception, parity, and error conditions, if necessary.

DMA and Interrupt Multiplexing

See the *Direct Memory Access (DMA)* chapter for information on DMA multiplexing. Several interrupts and DMA channels in the UART can be multiplexed.

NOTE: To operate in interrupt mode without using DMA channels, set the `UART_IMSK.ELSI` bit. This configuration redirects receive and transmit requests to the status interrupt output. The status interrupt goes directly to the SEC without going through the DMA controller.

Interrupt Masks

Each UART features a set of interrupt mask registers: `UART_IMSK`, `UART_IMSK_SET`, and `UART_IMSK_CLR`. The `UART_IMSK` register supports read/write operations. Writing ones to the `UART_IMSK_SET` register enables interrupts, writing ones to the `UART_IMSK_CLR` register disables them. Reads from either register return the enabled bits. This way, different interrupt service routines can control transmit, receive, and status interrupt requests independently and easily.

The UART module uses the `UART_IMSK` registers to enable requests for system handling of empty or full states of data registers. Unless polling is used as a means of action, the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` bits in this register are normally set.

Each UART module has three interrupt outputs. It uses one for transmission, one for reception, and one for reporting status events. The UART module routes transmit and receive requests through the DMA controller. The status request goes directly to the system event controller (SEC).

If the associated DMA channel is enabled, the request functions as a DMA request. If the DMA channel is disabled, it simply forwards the request to the SEC. A DMA channel must be associated with the UART module to enable transmit and receive interrupts. Otherwise, transmit and receive requests cannot be forwarded.

NOTE: To operate in interrupt mode without using DMA channels, set the `UART_IMSK.ELSI` bit. This configuration redirects receive and transmit requests to the status interrupt request output. The status interrupt goes directly to the SEC without going through the DMA controller.

Interrupt Servicing

Interrupt service routines (ISRs) perform UART writes and reads. Separate interrupt lines are provided for transmit, receive, and status. The `UART_IMSK` register group enables the independent interrupts individually. To enable UART transmit interrupts, set the `UART_CTL.EN` bit.

The ISRs evaluate the status bits in the `UART_STAT` register to determine the signaling interrupt source. The system event controller for the processor assigns and unmarks interrupts. The ISRs must clear the interrupt latches explicitly. To reduce interrupt frequency on the receive side in core mode, use the `UART_IMSK.ERFCI` status

interrupt as an alternative to the regular `UART_IMSK.ERBFI` receive interrupt. Hardware must ensure that at least two (if `UART_CTL.RFIT=0`) or four (if `UART_CTL.RFIT=1`) words are available in the receive buffer by the time the interrupt is requested.

Transmit Interrupts

The UART module uses the `UART_IMSK_SET.ETBEI` bit to enable transmit interrupt requests.

The `UART_THR` and `UART_TAIP` registers are the same physical register, and both affect the signaling of the `UART_STAT.TEMT`, `UART_STAT.TFI`, and `UART_STAT.THRE` bits similarly.

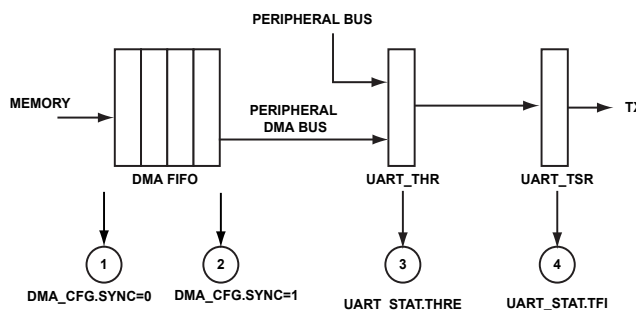


Figure 17-9: Transmit Interrupts

The UART module asserts the transmit request along with the `UART_STAT.THRE` bit, indicating that the transmit buffer is ready for new data. The `UART_STAT.THRE` bit resets to 1. When the `UART_IMSK_SET.ETBEI` bit is set, the UART module immediately issues an interrupt or DMA request. This way, no special handling of the first character is required when transmission of a string is initiated. Set the `UART_IMSK_SET.ETBEI` bit and let the interrupt service routine load the first character from memory and write it to the `UART_THR` register in the normal manner. ISRs can clear the `UART_IMSK.ETBEI` bit through the `UART_IMSK_CLR` register when the string transmission has completed.

Hardware clears the `UART_STAT.THRE` bit when new data is written to the `UART_THR` register. These write operations also clear the transmit interrupt request. However, they also initiate further transmission. If continued transmission is undesirable, the UART module can alternatively clear the transmit request through the `UART_IMSK_CLR.ETBEI` bit register. Transfers of data from the `UART_THR` register to the `UART_TSR` register reset this status flag in the `UART_STAT` register.

ISRs can interrogate the `UART_STAT.TEMT` bit to discover any ongoing transmission. The sticky counterpart of the `UART_STAT.TEMT` bit, `UART_STAT.TFI`, indicates when the transmit buffer has drained and can trigger a status interrupt. When data is pending in either one of these registers, the `UART_STAT.TEMT` flag is low. As soon as all data has left the `UART_TSR` register, the `UART_STAT.TEMT` bit goes high again and indicates that all pending transmit operations (including stop bits) have finished. Then, it is safe to disable the `UART_CTL.EN` bit or to three-state off-chip line drivers. Then, the UART module can generate an interrupt either through the status interrupt channel when the `UART_IMSK.ETFI` bit is set, or through the DMA controller when enabled by the `UART_IMSK.EDTPTI` bit.

When enabled by the `UART_IMSK.ETBEI` bit, the `UART_STAT.THRE` flag requests data along the peripheral command lines to the DMA controller (referred to as TXREQ). This signal is routed through the DMA controller. If the associated DMA channel is enabled, the TXREQ signal functions as a DMA request, otherwise the DMA controller simply forwards it to the SEC. Alternatively the `UART_IMSK.ETXS` bit can redirect the transmit interrupts to the UART status interrupt.

With interrupts disabled, the UART module can poll the status flags to determine when data is ready to move. Because polling is processor intensive, it is not typically used in real-time signal processing environments. Since read operations from `UART_STAT` registers have no side effects, different software threads can interrogate these registers without mutual impacts.

Polling the `SEC_SSTAT[n]` register without enabling the interrupts by the `SEC_CCTL[n]` register is an alternate method of operation to consider. Software can write up to two words into the `UART_THR` register before enabling the UART clock. As soon as the `UART_CTL.EN` bit is set, the UART module sends those two words.

Receive Interrupts

The UART module uses the `UART_IMSK_SET.ERBFI` bit to enable receive interrupt requests. If set, the `UART_STAT.DR` flag requests an interrupt on the dedicated RXREQ output, indicating that new data is available in the `UART_RBR` register. This signal is routed through the DMA controller. If the associated DMA channel is enabled, the RXREQ signal functions as a DMA request; otherwise the DMA controller simply forwards it to the SEC. Alternatively, if no DMA channel is assigned to the UART, the `UART_IMSK.ERXS` bit can redirect the receive interrupts to the UART status interrupt. When software reads the `UART_RBR` register, hardware clears the `UART_STAT.DR` bit again, which, in turn, clears the receive interrupt request.

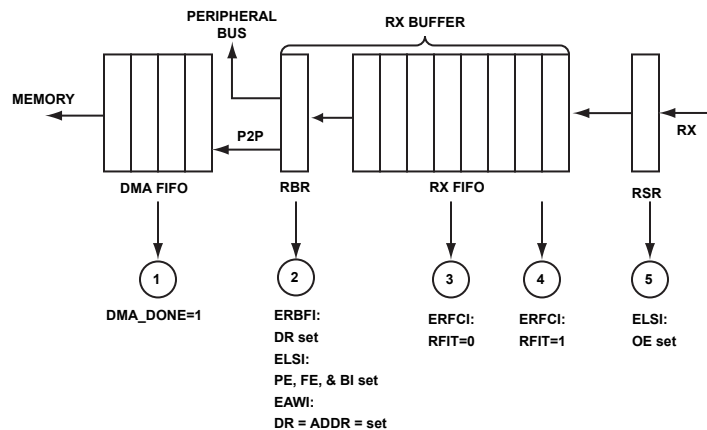


Figure 17-10: Receive Interrupts

Hardware updates the following:

- `UART_STAT.DR` bits
- `UART_STAT.ADDR` bits
- `UART_STAT.ASTKY` bits
- `UART_STAT.PE` bits

- `UART_STAT.FE` bits
- `UART_STAT.BI` bits
- `UART_RBR` register

The `UART_STAT.OE` bit is updated as soon as an overflow condition occurs (for example when a stop bit for a frame is received and the receive FIFO is full). When software does not read the `UART_RBR` register in time, the received data is protected from being overwritten by new data until software clears the `UART_STAT.OE` bit. Only the content of the `UART_RSR` register can be overwritten in the overrun case.

The UART module uses the `UART_STAT.RFCS` bit to monitor the state of the 8-deep receive FIFO. It uses the `UART_CTL.RFIT` bit to control the behavior of the buffer. If `UART_CTL.RFIT` is zero, the `UART_STAT.RFCS` bit is set when the receive buffer holds four or more words. If `UART_CTL.RFIT` is set, the `UART_STAT.RFCS` bit is set when the receive buffer holds seven or more words. Hardware clears the `UART_STAT.RFCS` bit when a core or DMA reads the `UART_RBR` register and when the buffer is flushed below the level of four (`UART_CTL.RFIT=0`) or seven (`UART_CTL.RFIT=1`). If the associated interrupt bit `UART_IMSK.ERFCI` is enabled, a status interrupt request is reported when the `UART_STAT.RFCS` bit is set.

If errors are detected during reception, an interrupt can be requested from the status interrupt output. This status interrupt request goes directly to the SEC. The bit enables status interrupt requests.

The controller detects the following error conditions, shown with their associated bits in the `UART_STAT` register.

- Overrun error (`UART_STAT.OE` bit)
- Parity error (`UART_STAT.PE` bit)
- Framing error or invalid stop bit (`UART_STAT.FE` bit)
- Break indicator (`UART_STAT.BI` bit)

Status Interrupts

The UART module uses status interrupt channels for the following purposes:

- Line status interrupt requests
- Flow control interrupt requests
- Receive FIFO threshold interrupt requests
- Transmission finished interrupt request

The UART module uses the `UART_IMSK.ELSI` bit to enable the line status interrupts. If set, the status interrupt request is asserted with one of the `UART_STAT.BI`, `UART_STAT.FE`, `UART_STAT.PE`, or `UART_STAT.OE` receive errors bits. A WIC operation to the `UART_STAT` register clears the error bits. Once all error conditions are cleared, the interrupt request deasserts.

The UART module uses the `UART_IMSK_SET.ERFCI` bit to enable the receive FIFO count interrupt. If set, a status interrupt request is generated when the `UART_STAT.RFCS` is active. The `UART_STAT.RFCS` bit

indicates a receive buffer threshold level. If the `UART_CTL.RFIT` bit is cleared, software can safely read two words out of the `UART_RBR` register by the time the `UART_STAT.RFCS` interrupt occurs.

If the `UART_CTL.RFIT` bit is set, software can safely read four words. The interrupt request and the `UART_STAT.RFCS` bit are cleared when the `UART_RBR` is read enough of times, so that the receive buffer drains below the threshold of two (`UART_CTL.RFIT=0`) or four (`UART_CTL.RFIT=1`). Because in DMA mode a status service routine may not be permitted to read `UART_RBR`, this interrupt is only recommended in core mode. In DMA mode, use this functionality for error recovery only.

The UART module uses the `UART_IMSK_SET.EDSSI` bit to enable the flow control interrupts. If active, a status interrupt is generated when the sticky `UART_STAT.SCTS` bit register is set, indicating that the `UART_CTS` input of the transmitter been reasserted. A WIC operation to the `UART_STAT.SCTS` bit clears the interrupt request.

The UART module uses the `UART_IMSK_SET.ETFI` bit to enable the transmission finished interrupt. If active, a status interrupt request is asserted when the `UART_STAT.TFI` bit is set. The `UART_STAT.TFI` is the sticky version of the `UART_STAT.TEMT` bit, indicating that a byte that started transmission has finished. A WIC operation to the `UART_STAT.TFI` bit clears the interrupt request.

Multi-Drop Bus Events

Several status bits and interrupt features in the `UART_STAT` and `UART_IMSK` registers facilitate efficient data handling in multi-drop bus mode. These features include the address (`UART_STAT.ADDR`) bit, the address sticky (`UART_STAT.ASTKY`) bit and the enable address word interrupt (`UART_IMSK.EAWI`). One of the key features of the multi-drop bus protocol is its address bit. The address bit signifies to the slaves that the master is transmitting an address word (all read it) or a data word (only the addressed slave reads its). The UART hardware provides an efficient method of handling the situation described with the use of `UART_STAT.ASTKY` bit.

NOTE: The UART module uses the `UART_STAT.ASTKY` bit in multi-drop bus mode to indicate when an address operation for a peripheral is occurring. The `UART_STAT.ASTKY` bit is a sticky version of the `UART_STAT.ADDR` bit. Hardware sets the bit whenever the `UART_STAT.ADDR` bit is set. Only software clears it with a WIC operation. With the `UART_STAT.ASTKY` bit set, words are received irrespective of the mode bit or address bit setting. With the `UART_STAT.ASTKY` bit cleared, only address words (mode bit =1) are received and words with mode bit =0 are ignored in MDB mode. This bit does not affect reception in non-MDB modes. (Words with mode bit =0 are not moved from the `UART_RSR` register to the receive FIFO.)

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

To detect Autobaud:

1. Ensure that the timer is disabled.

2. Configure the following bits: `UART_CTL.MOD = 00`, `UART_CTL.LOOP_EN = 1`, `UART_CTL.WLS = 11` (8-bit data), and `UART_CTL.EN = 1`
3. Configure the following bits: `TIMER_TMR[n]_CFG.TMODE = 1101`, `TIMER_TMR[n]_CFG.OUTDIS = 1`, `TIMER_TMR[n]_CFG.IRQMODE = 10` and enable the timer.
4. Send test data through the host device and wait for the timer interrupt and disable the timer.

The bit rate can be derived from the timer period register value according to the formula provided in the [Auto-baud Detection](#) section.

Using Common Initialization Steps

When using the core or the DMA to execute transfers, the following steps are common to all UART modes.

1. All UART signals are multiplexed and compete with other functions at pin level. First, program the port registers according to the guidelines in the [PORT](#) chapter.
2. Program the `UART_CLK` register. Refer to [Bit Rate Generation](#) topic.
3. Program the `UART_CTL` register and enable the UART clock.

Using Core Transfers

Write data into the `UART_THR` register, when the `UART_STAT.THRE` bit is set, to initiate a core transmit operation. If the `UART_STAT.DR` bit is set, received data can be read from the `UART_RBR` register.

Using DMA Transfers

1. Make sure that the `UART_IMSK.ETBEI` or the `UART_IMSK.ERBFI` bits are cleared before configuring the DMA.
2. Configure the dedicated DMA channel.
3. Set the `UART_IMSK.ETBEI` or `UART_IMSK.ERBFI` bits to start the transfer.

Using Interrupts

Each UART features three interrupt signal outputs.

1. Enable individual interrupts in the system event controller (SEC).
2. Register IRQ handlers.
3. Use the interrupts mask registers to enable specific IRQ events.

Setting Up Hardware Flow Control

1. Configure automatic or manual hardware flow control for the receiver through the `UART_CTL.ARTS` bit, or the transmitter through the `UART_CTL.ACTS` bit.
2. Configure `UART_CTS` and `UART_RTS` polarity through the `UART_CTL.FCPOL` bit.

ADSP-2156x UART Register Descriptions

UART (UART) contains the following registers.

Table 17-9: ADSP-2156x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

Clock Rate Register

The `UART_CLK` register divides the system clock (`SCLK`) down to the bit clock.

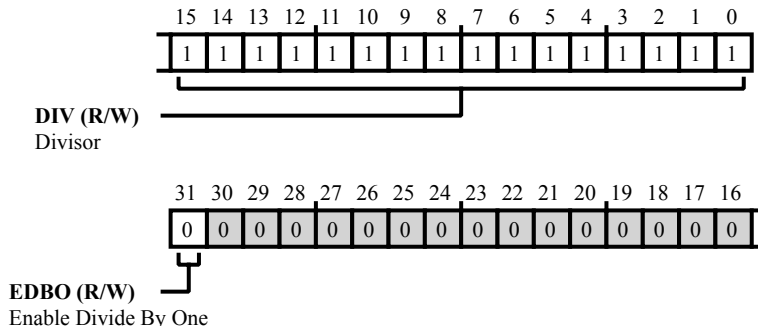


Figure 17-11: UART_CLK Register Diagram

Table 17-10: UART_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration				
31 (R/W)	EDBO	<p>Enable Divide By One.</p> <p>The <code>UART_CLK.EDBO</code> bit enables the bypassing of the divide-by-16 prescaler in bit clock generation. This functionality improves bit rate granularity, especially at high bit rates. Do not set this bit in IrDA mode.</p> <p>Note:</p> <p>Properly select the <code>SCLK</code> frequencies. Even multiples of bit rate decrease the error percentage.</p> <p>Setting the bit clock equal to the sample clock (<code>UART_CLK.EDBO=1</code>) improves the bit rate granularity and the bit clock matches with the bit rate of the communication partner.</p> <p>The disadvantage is that the power dissipation is higher and sample points are not always accurate. Therefore, use <code>UART_CLK.EDBO=1</code> mode only when bit rate accuracy is not acceptable in the <code>UART_CLK.EDBO=0</code> mode.</p> <p>The <code>UART_CLK.EDBO=1</code> mode is not intended to increase the speed of operation beyond the electrical limitations of the UART transfer protocol.</p> <table border="1" style="width: 100%; margin-top: 10px;"> <tr> <td style="text-align: center;">0</td> <td>Bit clock prescaler = 16</td> </tr> <tr> <td style="text-align: center;">1</td> <td>Bit clock prescaler = 1</td> </tr> </table>	0	Bit clock prescaler = 16	1	Bit clock prescaler = 1
0	Bit clock prescaler = 16					
1	Bit clock prescaler = 1					
15:0 (R/W)	DIV	<p>Divisor.</p> <p>The <code>UART_CLK.DIV</code> provides the divisor for the UART's clock bit rate calculation. The bit rate is defined by:</p> $\text{Bit Rate} = \text{SCLK} / (16^{(1-\text{EDBo})} \times \text{UART_CLK.DIV})$				

Control Register

The `UART_CTL` register provides enable and disable control for UART and IrDA mode of operation. It also provides UART line control, permitting selection of the format of received and transmitted character frames. Modem feature control also is available from this register, including partial modem functionality to allow for hardware flow control and loopback mode.

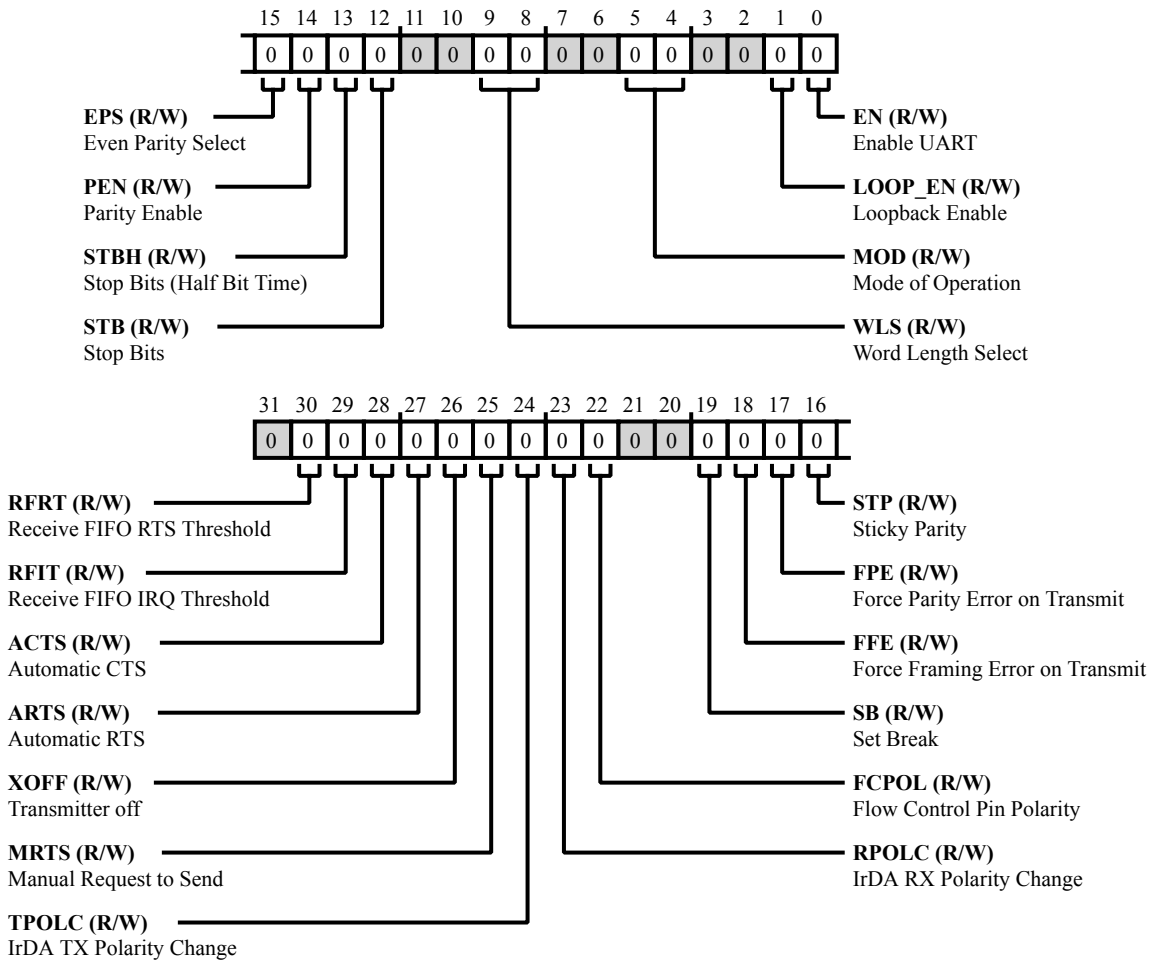


Figure 17-12: `UART_CTL` Register Diagram

Table 17-11: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W)	RFRT	Receive FIFO RTS Threshold. The <code>UART_CTL.RFRT</code> bit controls <code>UART_RTS</code> pin assertion and deassertion timing. This bit is ignored if <code>UART_CTL.ARTS</code> is cleared. If set, the <code>UART_RTS</code> pin is deasserted when the receive buffer already holds seven words and an eighth start bit is detected. It is reasserted when the FIFO contains seven words or less. If cleared, the <code>UART_RTS</code> pin is deasserted when the RX buffer already holds four words and a fifth start bit is detected. The <code>UART_RTS</code> pin is reasserted when the RX buffer contains no more than 4 words.
		0 Deassert RTS if RX FIFO word count > 4; assert if <= 4
		1 Deassert RTS if RX FIFO word count > 7; assert if <= 7
29 (R/W)	RFIT	Receive FIFO IRQ Threshold. The <code>UART_CTL.RFIT</code> bit controls the timing of the <code>UART_STAT.RFCS</code> bit. If <code>UART_CTL.RFIT</code> is cleared, the receive threshold is two. If <code>UART_CTL.RFIT</code> is set, the threshold is four words in the receive buffer.
		0 Set RFCS=1 if RX FIFO count >= 4
		1 Set RFCS=1 if RX FIFO count >= 7
28 (R/W)	ACTS	Automatic CTS. The <code>UART_CTL.ACTS</code> bit must be set to enable the <code>UART_CTS</code> input pin for <code>UART_TX</code> handshaking. If enabled, the <code>UART_STAT.CTS</code> bit holds the value (if <code>UART_CTL.FCPOL</code> is set) or complement value (if <code>UART_CTL.FCPOL</code> is cleared) of the <code>UART_CTS</code> input pin. The <code>UART_STAT.CTS</code> bit can be used to determine whether the external device is ready to receive data (if <code>UART_STAT.CTS</code> is set) or whether it is busy (if <code>UART_STAT.CTS</code> is cleared). If <code>UART_CTL.ACTS</code> is cleared, the <code>UART_TX</code> handshaking protocol is disabled, and the <code>UART_TX</code> line transmits data whenever there is data to send, regardless of the value of <code>UART_CTS</code> . Software can pause ongoing transmission by setting the <code>UART_CTL.XOFF</code> bit.
		0 Disable TX handshaking protocol
		1 Enable TX handshaking protocol
27 (R/W)	ARTS	Automatic RTS. The <code>UART_CTL.ARTS</code> bit must be set to enable the <code>UART_RTS</code> input pin for <code>UART_TX</code> handshaking. If set, the hardware guarantees a minimal <code>UART_RTS</code> pin deassertion pulse width of at least the number of data bits defined by the <code>UART_CTL.WLS</code> bit field. If cleared, the <code>UART_RTS</code> pin is not generated automatically by hardware. The <code>UART_RTS</code> pin can still be manually controlled by the <code>UART_CTL.MRTS</code> bit, and software is responsible for <code>UART_RTS</code> pulse width control (if needed).
		0 Disable RX handshaking protocol.
		1 Enable RX handshaking protocol.

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	XOFF	Transmitter off. The <code>UART_CTL.XOFF</code> bit (if set) turns off transmission (XOFF) by preventing the content of <code>THR</code> from being continued to <code>TSR</code> . When set, this bit turns on transmission (XON). The state of the <code>UART_CTL.XOFF</code> bit is ignored if the <code>UART_CTL.ACTS</code> bit is set.
		0 Transmission ON, if <code>ACTS</code> =0
		1 Transmission OFF, if <code>ACTS</code> =0
25 (R/W)	MRTS	Manual Request to Send. The <code>UART_CTL.MRTS</code> bit controls the state of the <code>UART_RTS</code> output pin when the <code>UART_CTL.ARTS</code> bit is cleared. When <code>UART_CTL.MRTS</code> is cleared, the UART deasserts the <code>UART_RTS</code> pin, signaling to the external device that the UART is not ready to receive. When <code>UART_CTL.MRTS</code> is set, the UART asserts the <code>UART_RTS</code> pin, signaling to the external device that the UART is ready to receive.
		0 Deassert RTS pin when <code>ARTS</code> =0
		1 Assert RTS pin when <code>ARTS</code> =0
24 (R/W)	TPOLC	IrDA TX Polarity Change. The <code>UART_CTL.TPOLC</code> bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the <code>UART_TX</code> pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the <code>UART_TX</code> pin idles low. In UART or MDB mode, it is NRZ.
		0 Active-low TX polarity setting
		1 Active-high TX polarity setting
23 (R/W)	RPOLC	IrDA RX Polarity Change. The <code>UART_CTL.RPOLC</code> bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the <code>UART_RX</code> pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the <code>UART_RX</code> pin idles low. In UART or MDB mode, it is NRZ.
		0 Active-low RX polarity setting
		1 Active-high RX polarity setting

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	FCPOL	Flow Control Pin Polarity. The <code>UART_CTL.FCPOL</code> bit selects the polarities of the <code>UART_CTS</code> and <code>UART_RTS</code> pins. When the <code>UART_CTL.FCPOL</code> bit is cleared, the <code>UART_RTS</code> and <code>UART_CTS</code> pins are active low, and the UART is halted when the <code>UART_RTS</code> and <code>UART_CTS</code> pin state is high. When <code>UART_CTL.FCPOL</code> bit is set, the <code>UART_RTS</code> and <code>UART_CTS</code> pins are active high, and the UART is halted when the <code>UART_RTS</code> and <code>UART_CTS</code> pin state is low.
		0 Active low CTS/RTS
		1 Active high CTS/RTS
19 (R/W)	SB	Set Break. If set, the <code>UART_CTL.SB</code> bit forces the <code>UART_TX</code> pin to low asynchronously, regardless of whether or not data is currently transmitted. This bit functions even when the UART clock is disabled. Because the <code>UART_TX</code> pin normally drives high, it can be used as a flag output pin, if the UART is not used. (For example, if <code>UART_CTL.TPOLC</code> is cleared, drive <code>UART_TX</code> pin low; or if <code>UART_CTL.TPOLC</code> is set, drive <code>UART_TX</code> pin high.)
		0 No force
		1 Force TX pin to 0
18 (R/W)	FFE	Force Framing Error on Transmit. The <code>UART_CTL.FFE</code> bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.
		0 Normal operation
		1 Force error
17 (R/W)	FPE	Force Parity Error on Transmit. The <code>UART_CTL.FPE</code> bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.
		0 Normal operation
		1 Force parity error

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	STP	Sticky Parity. The <code>UART_CTL.STP</code> bit controls whether the parity is generated by hardware based on the data bits or whether it is set to a fixed value. If this bit is cleared, the hardware calculates the parity bit value based on the data bits. Then, the <code>EPS</code> bit determines whether odd or even parity mode is chosen. If this bit is set, odd parity is used. That means that the total count of logical-1 data bits including the parity bit must be an odd value. Even parity is chosen by <code>UART_CTL.STP</code> cleared and <code>UART_CTL.EPS</code> set. Then, the count of logical-1 bits must be an even value. If the <code>UART_CTL.STP</code> bit is set, hardware parity calculation is disabled. In this case, the sent and received parity equals the inverted <code>UART_CTL.EPS</code> bit.
		0 No forced parity
		1 Force (Stick) parity to defined value (if <code>PEN=1</code>)
15 (R/W)	EPS	Even Parity Select.
		0 Odd parity
		1 Even parity
14 (R/W)	PEN	Parity Enable. The <code>UART_CTL.PEN</code> bit enables parity transmission and parity check. The <code>UART_CTL.PEN</code> bit inserts one additional bit between the most significant data bit and the first stop bit. The polarity of this so-called parity bit depends on data and the <code>UART_CTL.STP</code> and <code>UART_CTL.EPS</code> control bits. Both transmitter and receiver calculate the parity value. The receiver compares the received parity bit with the expected value and issues a parity error if they do not match. If the <code>UART_CTL.PEN</code> bit is cleared, the <code>UART_CTL.STP</code> and the <code>UART_CTL.EPS</code> bits are ignored.
		0 Disable
		1 Enable parity transmit and check
13 (R/W)	STBH	Stop Bits (Half Bit Time).
		0 0 half-bit-time stop bit
		1 1 half-bit-time stop bit
12 (R/W)	STB	Stop Bits. The <code>UART_CTL.STB</code> bit controls how many stop bits are appended to transmitted data.
		0 1 stop bit
		1 2 stop bits

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	WLS	Word Length Select. The <code>UART_CTL.WLS</code> field determines whether the transmitted and received UART word consists of 5, 6, 7, or 8 data bits.
		0 5-bit word
		1 6-bit word
		2 7-bit word
		3 8-bit word
5:4 (R/W)	MOD	Mode of Operation. The <code>UART_CTL.MOD</code> selects the UART operation mode (UMOD).
		0 UART mode
		1 MDB mode
		2 IrDA SIR mode
1 (R/W)	LOOP_EN	Loopback Enable. The <code>UART_CTL.LOOP_EN</code> bit enables UART loopback mode. When set, this bit disconnects the input of the receiver from the <code>UART_RX</code> pin, and internally redirects the transmit output to the receiver. The <code>UART_TX</code> pin remains active and continues to transmit data externally as well. Loopback mode also forces the <code>UART_RTS</code> pin to its deassertive state, disconnects the <code>UART_CTS</code> bit from the <code>UART_CTS</code> input pin, and directly connects the <code>UART_CTL.MRTS</code> bit to the <code>UART_STAT.CTS</code> bit. In loopback mode, setting the <code>UART_CTL.MRTS</code> bit sets the <code>UART_STAT.CTS</code> bit and enables the transmitter of the UART. Clearing the <code>UART_CTL.MRTS</code> bit clears the <code>UART_STAT.CTS</code> bit and disables the transmitter of the UART.
		0 Disable
		1 Enable
0 (R/W)	EN	Enable UART. The <code>UART_CTL.EN</code> enables the UART clocks. This bit also resets the state machine and control registers when cleared. Using this bit to disable the UART -- when not used -- reduces power consumption.
		0 Disable
		1 Enable

Interrupt Mask Register

The `UART_IMSK` register indicates the interrupt mask status (unmasked, if set, or masked, if cleared) of the UART status interrupt requests. This register is not a data register. Instead, it is controlled by the `UART_IMSK_SET` and `UART_IMSK_CLR` register pair. Writing ones to the `UART_IMSK_SET` register enables (unmasks) interrupt requests, and writing ones to the `UART_IMSK_CLR` register disables (masks) them. Reads from either register return the enabled bits.

The `UART_IMSK` register is used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` bits are normally set. Setting this register without enabling system DMA causes the UART to notify the processor of the data inventory state using interrupts. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present.

Each UART features three separate interrupt channels to handle the data transmit, data receive, and line status events independently, regardless of whether DMA is enabled or not. If no DMA channels are assigned to the UART, set the `UART_IMSK.ELSI` bit to reroute the transmit and receive interrupts to the status interrupt request output.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available for receive and transmit operations. Line error handling can be configured independently from the receive or transmit setup.

The DMA of the UART is enabled by first setting up the system DMA control registers and then enabling the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` interrupts. This configuration is because the interrupt request lines double as DMA request lines. Depending on whether DMA is enabled or not, upon receiving these requests, the DMA control unit either generates a direct memory access or passes the UART interrupt on to the system interrupt handling unit(s). However, the error interrupt for the UART goes directly to the system interrupt handling unit(s), bypassing the DMA unit completely.

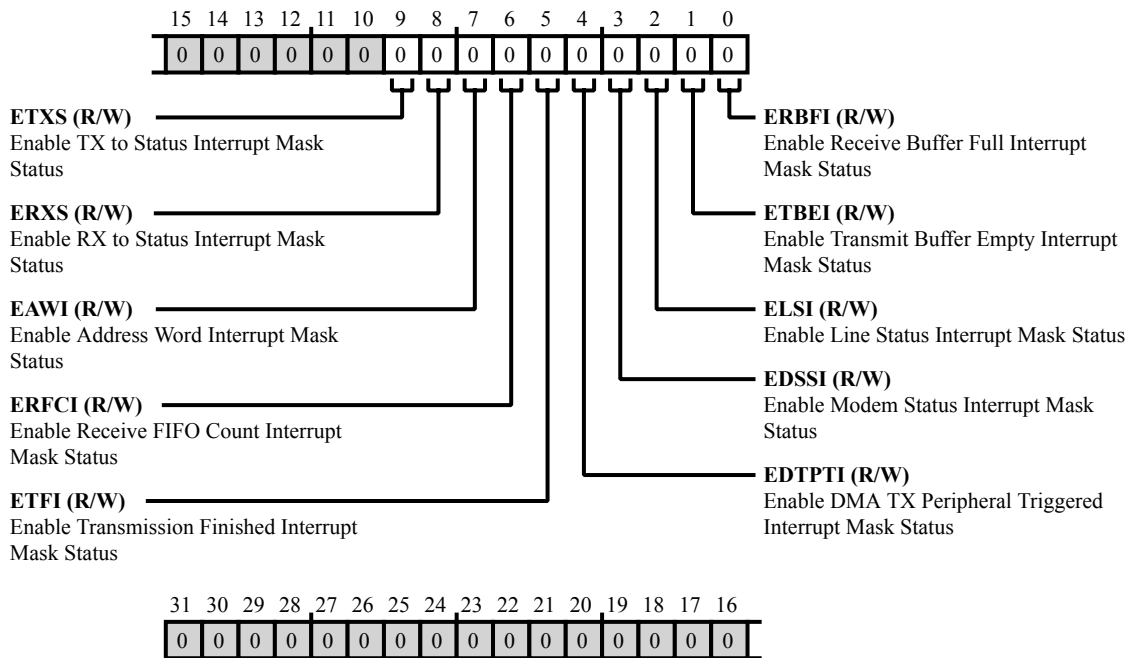


Figure 17-13: UART_IMSK Register Diagram

Table 17-12: UART_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	ETXS	Enable TX to Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ETXS bit indicates re-direction of the TX interrupt requests to status interrupt output. If cleared, TX interrupt requests are routed to normal interrupt outputs.
		0 Interrupt is masked
		1 Interrupt is unmasked
8 (R/W)	ERXS	Enable RX to Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ERXS bit indicates re-direction of RX interrupt requests to status interrupt output. If cleared, RX interrupt requests are routed to normal interrupt outputs.
		0 Interrupt is masked
		1 Interrupt is unmasked

Table 17-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	EAWI	Enable Address Word Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.EAWI</code> bit indicates generation of a status interrupt request when an Address word in MDB-mode is present in the <code>UART_RBR</code> . A received word is an address word if the <code>UART_STAT.ADDR</code> bit is set.
		0 Interrupt is masked
		1 Interrupt is unmasked
6 (R/W)	ERFCI	Enable Receive FIFO Count Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.ERFCI</code> bit indicates enabling of the receive buffer threshold interrupt request if signaled by the <code>UART_STAT.RFCS</code> bit. Read the <code>UART_RBR</code> register sufficient times to clear the interrupt request.
		0 Interrupt is masked
		1 Interrupt is unmasked
5 (R/W)	ETFI	Enable Transmission Finished Interrupt Mask Status. If set (interrupt unmasked) the <code>UART_IMSK.ETFI</code> bit indicates enabling of interrupt generation on the status interrupt channel when the transmit buffer register, the transmit address register, and the transmit shift register are all empty as indicated by the <code>UART_STAT.TFI</code> . The <code>UART_IMSK.ETFI</code> interrupt can be used to avoid expensive polling of the <code>UART_STAT.TEMT</code> bit, when the UART clock or line drivers should be disabled after transmission has completed. WIC the <code>UART_STAT.TFI</code> bit to clear the interrupt request. In DMA operation, the <code>UART_IMSK.ETFI</code> bits functionality might be preferred.
		0 Interrupt is masked
		1 Interrupt is unmasked
4 (R/W)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.EDTPTI</code> bit indicates enabling of the DMA completion interrupt request to be delayed until the data has left the UART completely. This bit is required for DMA transmit operation only. If set, the UART can generate a DMA interrupt request by the time the <code>UART_STAT.TEMT</code> bit goes high after the last DMA data word is transmitted. When <code>UART_IMSK.EDTPTI</code> is set, usually the <code>DMA_CFG.INT</code> field is cleared to 00 in a STOP mode DMA. This set up suppresses the normal completion interrupt request, and the <code>UART_STAT.TEMT</code> event is signaled through the DMA controller and triggers the DMA interrupt. If both (<code>DMA_CFG.INT</code> not 00 and <code>UART_IMSK.EDTPTI</code> set), two interrupts are requested at the end of a STOP mode DMA.
		0 Interrupt is masked
		1 Interrupt is unmasked

Table 17-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	EDSSI	Enable Modem Status Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.EDSSI</code> bit indicates enabling of a modem status interrupt request on the same status interrupt channel when the <code>UART_STAT.SCTS</code> bit is set. This indicates <code>UART_CTS</code> pin re-assertion. Write-1-to-clear (W1C) the <code>UART_STAT.SCTS</code> bit to clear the interrupt request.
		0 Interrupt is masked
		1 Interrupt is unmasked
2 (R/W)	ELSI	Enable Line Status Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.ELSI</code> bit indicates that redirection of TX and RX interrupt requests to the status interrupt output of the UART by OR'ing them with the <code>UART_STAT.OE</code> , <code>UART_STAT.PE</code> , <code>UART_STAT.FE</code> , and <code>UART_STAT.BI</code> interrupt requests. Set this bit when no DMA channel is associated with the UART. Enabling <code>UART_IMSK.ELSI</code> disables the RX/TX interrupt channels and negates the <code>UART_IMSK.EDTPTI</code> bit.
		0 Interrupt is masked
		1 Interrupt is unmasked
1 (R/W)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.ETBEI</code> bit indicates generation of a TX interrupt request if the <code>UART_STAT.THRE</code> bit is set.
		0 Interrupt is masked
		1 Interrupt is unmasked
0 (R/W)	ERBFI	Enable Receive Buffer Full Interrupt Mask Status. If set (interrupt unmasked), the <code>UART_IMSK.ERBFI</code> indicates generation of an RX interrupt request if the <code>UART_STAT.DR</code> bit is set.
		0 Interrupt is masked
		1 Interrupt is unmasked

Interrupt Mask Clear Register

The `UART_IMSK` indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the `UART_IMSK_SET` and `UART_IMSK_CLR` register pair. Writing ones to `UART_IMSK_SET` enables (unmasks) interrupt requests, and writing ones to `UART_IMSK_CLR` disables (masks) them. Reads from either register return the enabled bits. For more information, see the `UART_IMSK` register description.

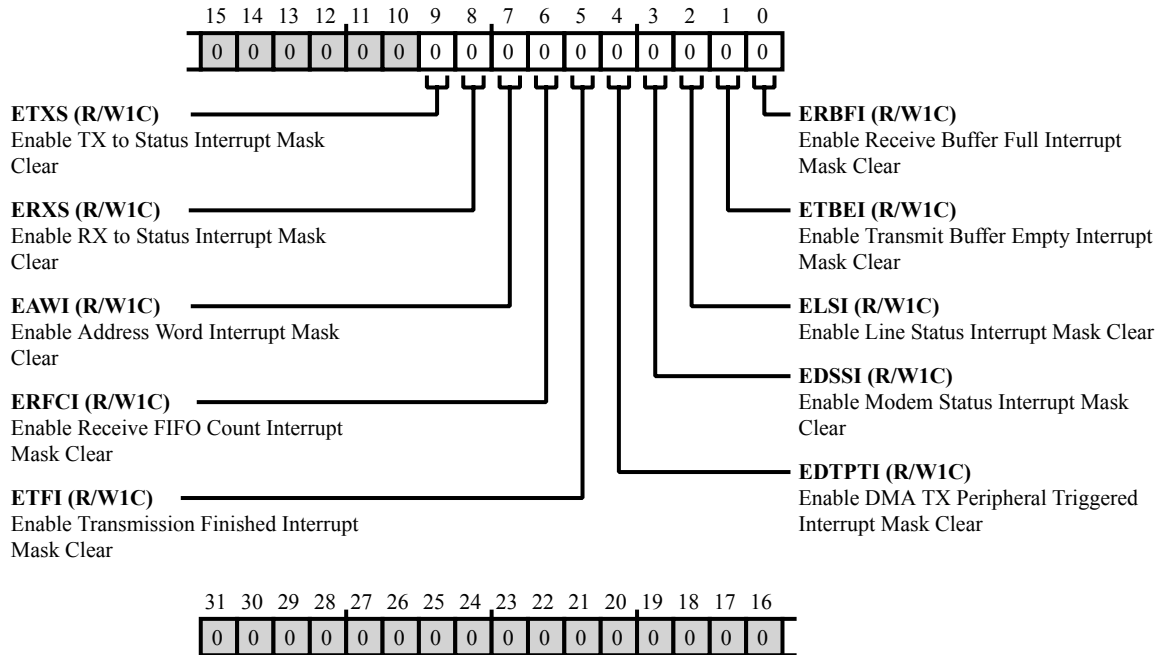


Figure 17-14: `UART_IMSK_CLR` Register Diagram

Table 17-13: `UART_IMSK_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W1C)	ETXS	Enable TX to Status Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
8 (R/W1C)	ERXS	Enable RX to Status Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
7 (R/W1C)	EAWI	Enable Address Word Interrupt Mask Clear.
		0 No action
		1 Mask interrupt

Table 17-13: UART_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	ERFCI	Enable Receive FIFO Count Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
5 (R/W1C)	ETFI	Enable Transmission Finished Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
4 (R/W1C)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
3 (R/W1C)	EDSSI	Enable Modem Status Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
2 (R/W1C)	ELSI	Enable Line Status Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
1 (R/W1C)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Clear.
		0 No action
		1 Mask interrupt
0 (R/W1C)	ERBFI	Enable Receive Buffer Full Interrupt Mask Clear.
		0 No action
		1 Mask interrupt

Interrupt Mask Set Register

The `UART_IMSK` indicates interrupt request mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the `UART_IMSK_SET` and `UART_IMSK_CLR` register pair. Writing ones to `UART_IMSK_SET` enables (unmasks) interrupt requests, and writing ones to `UART_IMSK_CLR` disables (masks) them. Reads from either register return the enabled bits. For more information, see the `UART_IMSK` register description.

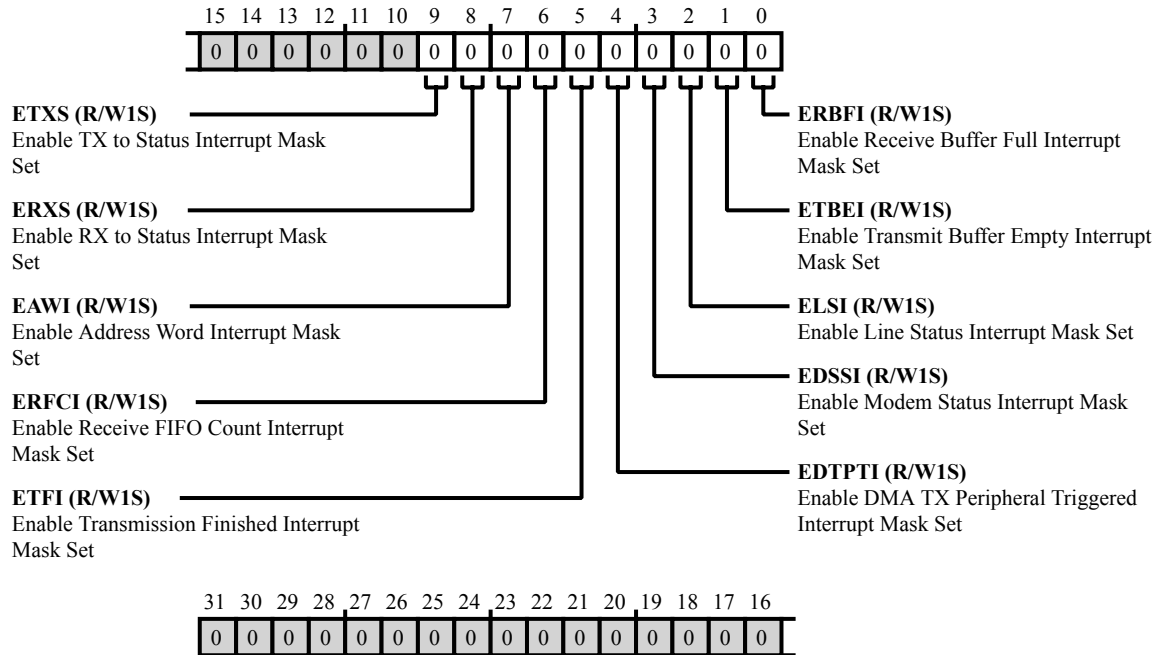


Figure 17-15: UART_IMSK_SET Register Diagram

Table 17-14: UART_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W1S)	ETXS	Enable TX to Status Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
8 (R/W1S)	ERXS	Enable RX to Status Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
7 (R/W1S)	EAWI	Enable Address Word Interrupt Mask Set.
		0 No action
		1 Unmask interrupt

Table 17-14: UART_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1S)	ERFCI	Enable Receive FIFO Count Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
5 (R/W1S)	ETFI	Enable Transmission Finished Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
4 (R/W1S)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
3 (R/W1S)	EDSSI	Enable Modem Status Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
2 (R/W1S)	ELSI	Enable Line Status Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
1 (R/W1S)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Set.
		0 No action
		1 Unmask interrupt
0 (R/W1S)	ERBFI	Enable Receive Buffer Full Interrupt Mask Set.
		0 No action
		1 Unmask interrupt

Receive Buffer Register

The read-only `UART_RBR` register is the UART's receive buffer. It is updated when there is pending data in the receive FIFO. Newly available data is signaled by the `UART_STAT.DR` bit.

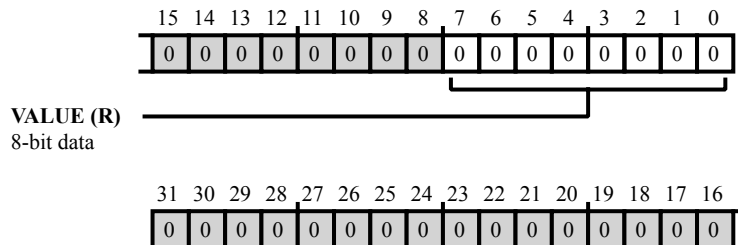


Figure 17-16: UART_RBR Register Diagram

Table 17-15: UART_RBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	8-bit data.

Receive Shift Register

The read only `UART_RSR` register which returns the content of the UART's receive shift register.

The frame data is moved into this shift register after polarity inversion, if any (including the native polarity inversion in the IrDA case).

In the case of the longest frame (MDB, with parity mode, and 8 bit data word-length), the start bit may be shifted out and not available for reading at the end of the frame reception. This register is NOT reset at the start of frame. If read, in the middle of a frame reception, data corresponding the previous frame may not have entirely shifted out (for example, the read data that have been read may NOT correspond entirely to the frame being received).

Because the UART is receiving only 1 stop bit, the `UART_RSR` contains only 1 stop bit even if more than one stop bit is present in the actual transfer. This register may be considered as storing the 10 most recently received bits (taking into consideration the stop bit receive limitation above).

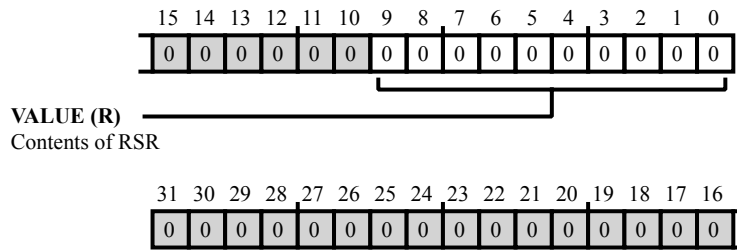


Figure 17-17: UART_RSR Register Diagram

Table 17-16: UART_RSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	VALUE	Contents of RSR.

Receive Counter Register

The `UART_RXCNT` register returns the content of 16-bit counter in the UART receiver. This count is used for baud rate clock generation (the lower [15:0] is the count data).

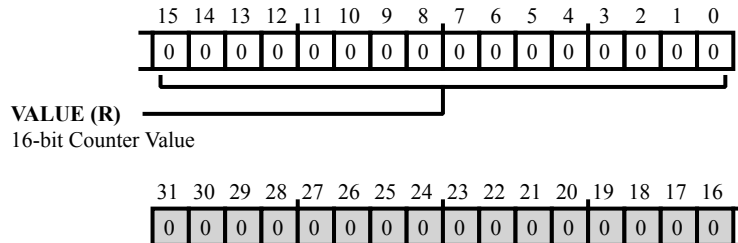


Figure 17-18: UART_RXCNT Register Diagram

Table 17-17: UART_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

Scratch Register

The `UART_SCR` registers contain 8-bit scratch pad data. These registers are used for general purpose data storage and do not control the UART hardware in any way.

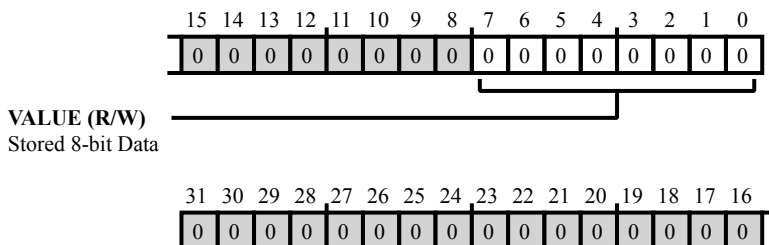


Figure 17-19: UART_SCR Register Diagram

Table 17-18: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Stored 8-bit Data.

Status Register

The `UART_STAT` register contains the UART line status and UART modem status, as indicated by the current states of the UART's `UART_CTS` pin and internal receive buffers. Writes to this register can perform write-one-to-clear (W1C) operations on most status bits. Reading this register has no side effects.

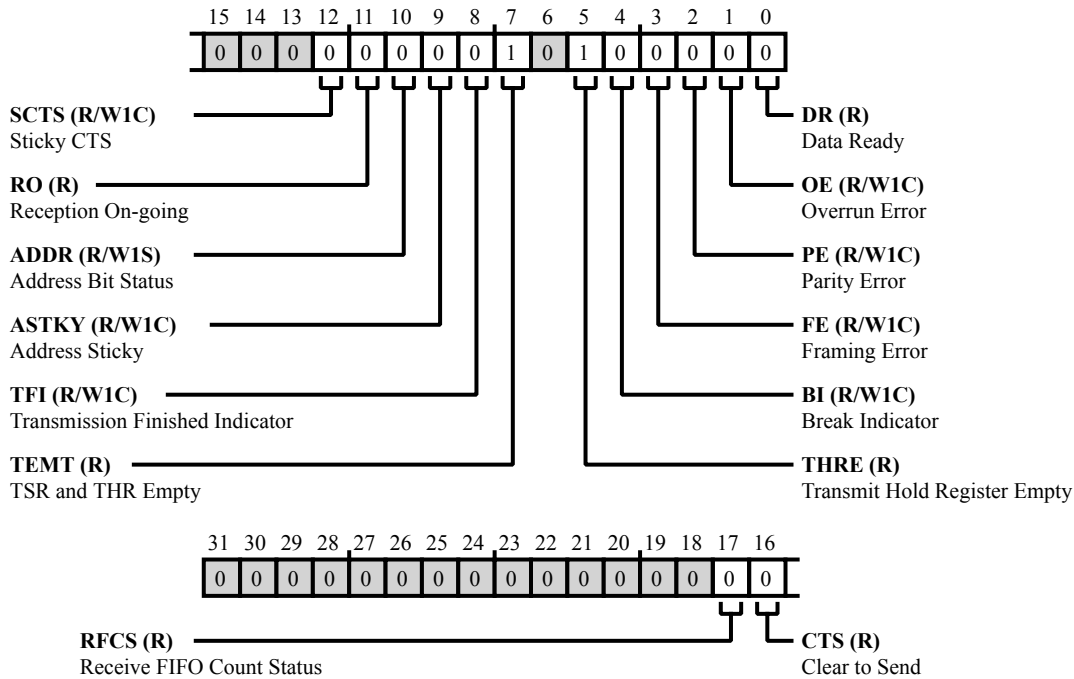


Figure 17-20: `UART_STAT` Register Diagram

Table 17-19: `UART_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	RFCS	<p>Receive FIFO Count Status.</p> <p>The <code>UART_STAT.RFCS</code> bit is set when the receive buffer holds more or equal entries than a certain threshold. The threshold is controlled by the <code>UART_CTL.RFIT</code> bit. If <code>UART_CTL.RFIT</code> is cleared, the threshold is four entries. If <code>UART_CTL.RFIT</code> is set, the threshold is seven entries. The <code>UART_STAT.RFCS</code> bit is cleared when the <code>UART_RBR</code> register is read sufficient times until the buffer is drained below the threshold. The <code>UART_STAT.RFCS</code> bit can trigger a status interrupt request if enabled by the <code>UART_IMSK_SET.ERFCI</code> bit.</p>
		0 RX FIFO has less than 4 (7) entries when RFIT=0 (1)
		1 RX FIFO has at least 4 (7) entries when RFIT=0 (1)

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/NW)	CTS	Clear to Send. The <code>UART_STAT.CTS</code> bit holds the value (if <code>UART_CTL.FCPOL</code> set) or the complement value (if <code>UART_CTL.FCPOL</code> cleared) of the <code>UART_CTS</code> input pin. The <code>UART_CTL.ACTS</code> bit must be set to enable this feature. The core can read the value of the <code>UART_STAT.CTS</code> bit to determine whether the external device is ready to receive (<code>UART_STAT.CTS</code> set) or if it is busy (<code>UART_STAT.CTS</code> cleared). If <code>UART_CTL.ACTS</code> is cleared, the <code>UART_TX</code> handshaking protocol is disabled, and the UART transmits data as long as there is data to transmit, regardless of the value of <code>UART_STAT.CTS</code> . When <code>UART_CTL.ACTS</code> is cleared, the software can pause transmission temporarily by setting the <code>XOFF</code> bit. Note that in loopback mode (<code>UART_CTL.LOOP_EN</code> set), the <code>UART_STAT.CTS</code> bit is disconnected from the <code>UART_CTS</code> input pin. Instead, the bit is directly connected to the <code>UART_CTL.MRTS</code> bit.
		0 Not clear to send (External device not ready to receive)
		1 Clear to send (External device ready to receive)
12 (R/W1C)	SCTS	Sticky CTS. The <code>UART_STAT.SCTS</code> bit is a sticky bit that is set when <code>UART_STAT.CTS</code> transitions from 0 to 1. The <code>UART_STAT.SCTS</code> bit is cleared by software with a W1C operation. This bit can trigger a line status interrupt request if enabled by the <code>UART_IMSK_SET.EDSSI</code> bit.
		0 CTS has not transitioned from low to high
		1 CTS has transitioned from low to high
11 (R/NW)	RO	Reception On-going.
		0 No data reception in progress
		1 Data reception in progress
10 (R/W1S)	ADDR	Address Bit Status. The <code>UART_STAT.ADDR</code> bit is used to mirror the address bit of the word in <code>UART_RBR</code> in multi-drop bus protocol, and is enabled only in MDB mode. The <code>UART_STAT.ADDR</code> bit is updated by hardware upon detecting a received word with the address bit in <code>UART_RBR</code> set or cleared. Additionally, software can set the <code>ADDR</code> bit with a write-1-to-set (W1S) operation.
		0 Address bit is low
		1 Address bit is high

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W1C)	ASTKY	Address Sticky. The <code>UART_STAT.ASTKY</code> bit is used in multi-drop bus mode to indicate whether a peripheral is currently being addressed. This bit is a sticky version of the <code>UART_STAT.ADDR</code> bit and is set by hardware when setting the <code>UART_STAT.ADDR</code> bit. The <code>UART_STAT.ASTKY</code> bit can only be cleared by software with a write-one-to-clear (W1C) operation. With the <code>UART_STAT.ASTKY</code> bit set, words will be received irrespective of the <code>UART_CTL.MOD</code> bit or <code>UART_STAT.ADDR</code> bit selection. With the <code>UART_STAT.ASTKY</code> bit cleared, only address words (<code>UART_CTL.MOD</code> bit set) will be received and words with <code>UART_CTL.MOD</code> bit cleared are ignored (not moved from the RSR to the RX FIFO) in MDB mode. The <code>UART_STAT.ASTKY</code> bit does not affect reception in non-MDB modes.
		0 ADDR bit has not been set
		1 ADDR bit has been set
8 (R/W1C)	TFI	Transmission Finished Indicator. The <code>UART_STAT.TFI</code> bit is a sticky version of the <code>UART_STAT.TEMT</code> bit. While <code>UART_STAT.TEMT</code> is automatically cleared by hardware when new data is written to the <code>UART_THR</code> register, the sticky <code>UART_STAT.TFI</code> bit remains set, until it is cleared by software (W1C). The <code>UART_STAT.TFI</code> bit enables more flexible transmit interrupt request timing.
		0 TEMT did not transition from 0 to 1
		1 TEMT transition from 0 to 1
7 (R/NW)	TEMT	TSR and THR Empty. The <code>UART_STAT.TEMT</code> bit indicates that the <code>UART_THR</code> and <code>UART_TAIP</code> registers and the <code>UART_TSR</code> register are empty. In this case, the program is permitted to write to the <code>UART_THR</code> and <code>UART_TAIP</code> registers twice without losing data. The <code>UART_STAT.TEMT</code> bit can also be used as indicator that pending UART transmission is completed. At that time, it is safe to disable the <code>UART_CTL.EN</code> bit or to three-state the off-chip line driver.
		0 Not empty TSR/THR
		1 TSR/THR Empty

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	THRE	Transmit Hold Register Empty. The <code>UART_STAT.THRE</code> bit indicates that the UART transmit channel is ready for new data and software can write to the <code>UART_THR</code> and <code>UART_TAIP</code> registers. Writes to the <code>UART_THR</code> and <code>UART_TAIP</code> registers clear the <code>UART_STAT.THRE</code> . The bit is set again when the <code>UART_THR</code> and <code>UART_TAIP</code> registers are empty and ready to accept data.
		0 Not empty THR/TAIP
		1 Empty THR/TAIP
4 (R/W1C)	BI	Break Indicator. The <code>UART_STAT.BI</code> bit indicates that the first stop bit is sampled low and the entire <u>data word</u> , including parity bit, consists of low bits only. (This condition indicates that <code>UART_RX</code> was held low for more than the maximum word length.) The <code>UART_STAT.BI</code> bit is updated simultaneously with the <code>UART_STAT.DR</code> bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the <code>UART_RBR</code> register. The bit is sticky and can be cleared by W1C operations.
		0 No break interrupt
		1 Break interrupt this indicates UARTxRX was held low (RPOLC=0) / high (RPOLC=1) for more than the maximum word length
3 (R/W1C)	FE	Framing Error. The <code>UART_STAT.FE</code> bit indicates that the first stop bit is sampled. This bit is updated simultaneously with the <code>UART_STAT.DR</code> bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the <code>UART_RBR</code> register. The <code>UART_STAT.FE</code> bit is sticky and can be cleared by W1C operations. Note that invalid stop bits can be simulated by setting the <code>UART_CTL.FFE</code> bit.
		0 No error
		1 Invalid stop bit error
2 (R/W1C)	PE	Parity Error. The <code>UART_STAT.PE</code> bit indicates that the received parity bit does not match the expected value. This bit is updated simultaneously with the <code>UART_STAT.DR</code> bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the <code>UART_RBR</code> register. The <code>UART_STAT.PE</code> bit is sticky and can be cleared by W1C operations. Note that invalid parity bits can be simulated by setting the <code>UART_CTL.FPE</code> bit.
		0 No parity error
		1 Parity error

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	OE	<p>Overrun Error.</p> <p>The <code>UART_STAT.OE</code> bit indicates that further data is received while the internal receive buffer was full. This bit is set when sampling the stop bit of the sixth data word. To avoid overruns, read the <code>UART_RBR</code> register in time. In DMA receive mode, overruns are very unlikely to happen ever. After an overrun occurs, the <code>UART_RBR</code> and receive FIFO are protected from being overwritten by new data until the <code>UART_STAT.OE</code> bit is cleared by software. The content of the <code>UART_RSR</code> register is lost as soon as the overrun occurs. The <code>UART_STAT.OE</code> bit is sticky and can be cleared by W1C operations.</p>
		0 No overrun
		1 Overrun error
0 (R/NW)	DR	<p>Data Ready.</p> <p>The <code>UART_STAT.DR</code> bit indicates that data is available in the receiver and can be read from the <code>UART_RBR</code> register. The bit is set by hardware when the receiver detects the first valid stop bit. The bit is cleared by hardware when the <code>UART_RBR</code> register is read.</p>
		0 No new data
		1 New data in RBR

Transmit Address/Insert Pulse Register

The `UART_TAIP` register and the `UART_THR` register share the same physical register, but `UART_TAIP` has different effect than the `UART_THR` register when `UART_TAIP` is written to in MDB and UART modes.

In MDB mode, data written to the `UART_TAIP` register is transmitted as an address frame (as with the `UART_CTL.MOD` bit set).

In UART mode, a write to `UART_TAIP` causes a pulse of value `UART_TAIP` [7] for a duration of `UART_TAIP` [6:0] x bit time. (There is additional inversion if the `UART_CTL.TPOLC` bit is set).

Bit time is defined by the `UART_CLK` register. The transmission of the pulse is followed by stop bit transmission as specified by the `UART_CTL.STB` and `UART_CTL.STBH` bits. This could be used for supporting line break command and inter-frame gap.

In IrDA mode, writes to `UART_TAIP` is treated the same as writes to `UART_THR`.

Accesses to the `UART_TAIP` register have the same affects as the `UART_THR` register with respect to the `UART_STAT.THRE`, `UART_STAT.TEMT`, and `UART_STAT.TFI` flags.

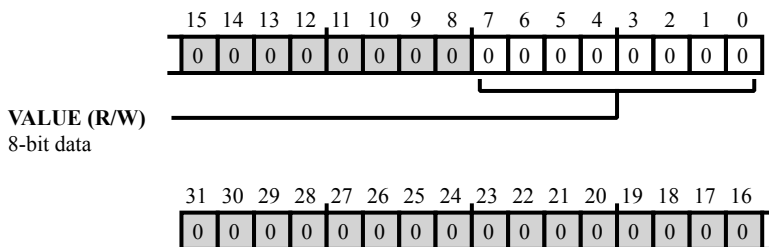


Figure 17-21: `UART_TAIP` Register Diagram

Table 17-20: `UART_TAIP` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8-bit data.

Transmit Hold Register

The write-only `UART_THR` register is the UART’s transmit buffer. The `UART_STAT.THRE` bit indicates whether data can be written to `UART_THR`. Writes to this register automatically propagate to the internal `UART_TSR` register as soon as `UART_TSR` is ready. Then, transmit operation is initiated immediately.

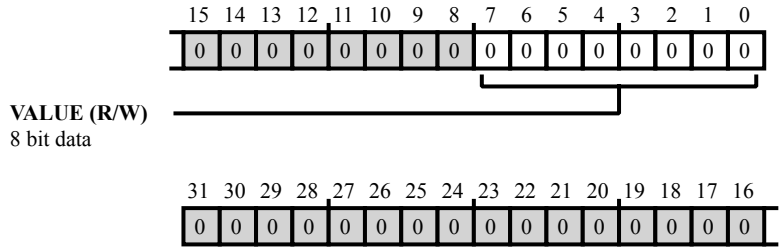


Figure 17-22: UART_THR Register Diagram

Table 17-21: UART_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8 bit data.

Transmit Shift Register

The read only `UART_TSR` register which returns the content of the UART’s transmit shift register.

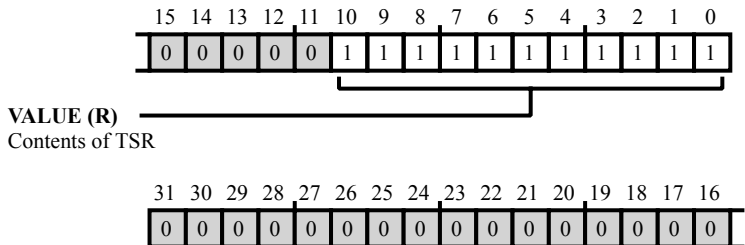


Figure 17-23: UART_TSR Register Diagram

Table 17-22: UART_TSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Contents of TSR.

Transmit Counter Register

The `UART_TXCNT` read only register returns the content of 16-bit counter in the UART transmitter. This count is used for baud rate clock generation (the lower [15:0] is the count data).

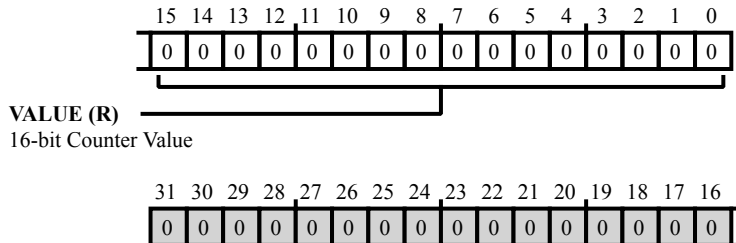


Figure 17-24: UART_TXCNT Register Diagram

Table 17-23: UART_TXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

18 General-Purpose Timer (TIMER)

The general-purpose timer (GP Timer) module serves as a collection of system timers that support various system-level functions. These functions include:

- Synchronized PWM waveform output capability
- External signal capture
- External event count
- General time-base functionality

Additionally, interrupt requests can be generated upon completion of timer events. Moreover, GP timers can act both as trigger masters and trigger slaves.

GP Timer Features

Each timer can be individually configured in any of these modes:

- Pin interrupt capture mode
- Windowed watchdog mode
- Pulse-width count and capture (WDTH_CAP) mode
- External Event (EXT_CLK) mode
- Pulse-width modulation (PWM_OUT) mode

Other features include:

- Synchronous operation
- Consistent management of period and pulse width values
- Autobaud detection for UART module (where available)
- Graceful bit pattern termination when stopping
- Support for center-aligned PWM patterns
- Error detection on implausible pattern values

- All read and write accesses to 32-bit registers are atomic
- Every timer has its dedicated interrupt request output

ADSP-2156x TIMER Register List

The General-Purpose Timer block (TIMER) provides timers that may be used for external event capture and measurement, system timing, and PWM waveform generation. A set of registers governs TIMER operations. For more information on TIMER functionality, see the TIMER register descriptions.

Table 18-1: ADSP-2156x TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

ADSP-2156x TIMER Interrupt List

Table 18-2: ADSP-2156x TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
22	TIMER0_TMR00	TIMER0 Timer 0	Level	
23	TIMER0_TMR01	TIMER0 Timer 1	Level	
24	TIMER0_TMR02	TIMER0 Timer 2	Level	
25	TIMER0_TMR03	TIMER0 Timer 3	Level	
74	TIMER0_TMR04	TIMER0 Timer 4	Level	
75	TIMER0_TMR05	TIMER0 Timer 5	Level	
76	TIMER0_TMR06	TIMER0 Timer 6	Level	
77	TIMER0_TMR07	TIMER0 Timer 7	Level	
78	TIMER0_TMR08	TIMER0 Timer 8	Level	
79	TIMER0_TMR09	TIMER0 Timer 9	Level	
80	TIMER0_STAT	TIMER0 Status	Level	

ADSP-2156x TIMER Trigger List

Table 18-3: ADSP-2156x TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
3	TIMER0_TMR00_MST	TIMER0 Timer 0	Edge
4	TIMER0_TMR01_MST	TIMER0 Timer 1	Edge
5	TIMER0_TMR02_MST	TIMER0 Timer 2	Edge
6	TIMER0_TMR03_MST	TIMER0 Timer 3	Edge
7	TIMER0_TMR04_MST	TIMER0 Timer 4	Edge
8	TIMER0_TMR05_MST	TIMER0 Timer 5	Edge
9	TIMER0_TMR06_MST	TIMER0 Timer 6	Edge
10	TIMER0_TMR07_MST	TIMER0 Timer 7	Edge
11	TIMER0_TMR08_MST	TIMER0 Timer 8	Edge
12	TIMER0_TMR09_MST	TIMER0 Timer 9	Edge

Table 18-4: ADSP-2156x TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	TIMER0_TMR00_SLV0	TIMER0 Timer 0	Pulse
1	TIMER0_TMR00_SLV1	TIMER0 Timer 0	Pulse

Table 18-4: ADSP-2156x TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR01_SLV0	TIMER0 Timer 1	Pulse
3	TIMER0_TMR01_SLV1	TIMER0 Timer 1	Pulse
4	TIMER0_TMR02_SLV0	TIMER0 Timer 2	Pulse
5	TIMER0_TMR02_SLV1	TIMER0 Timer 3	Pulse
6	TIMER0_TMR03_SLV0	TIMER0 Timer 3	Pulse
7	TIMER0_TMR03_SLV1	TIMER0 Timer 3	Pulse
8	TIMER0_TMR04_SLV0	TIMER0 Timer 4	Pulse
9	TIMER0_TMR04_SLV1	TIMER0 Timer 4	Pulse
10	TIMER0_TMR05_SLV0	TIMER0 Timer 5	Pulse
11	TIMER0_TMR05_SLV1	TIMER0 Timer 5	Pulse
12	TIMER0_TMR06_SLV0	TIMER0 Timer 6	Pulse
13	TIMER0_TMR06_SLV1	TIMER0 Timer 6	Pulse
14	TIMER0_TMR07_SLV0	TIMER0 Timer 7	Pulse
15	TIMER0_TMR07_SLV1	TIMER0 Timer 7	Pulse
16	TIMER0_TMR08_SLV0	TIMER0 Timer 8	Pulse
17	TIMER0_TMR08_SLV1	TIMER0 Timer 8	Pulse
18	TIMER0_TMR09_SLV0	TIMER0 Timer 9	Pulse
19	TIMER0_TMR09_SLV1	TIMER0 Timer 9	Pulse

Timer Block Diagram

The Timer block diagram figure shows all of the possible clock sources.

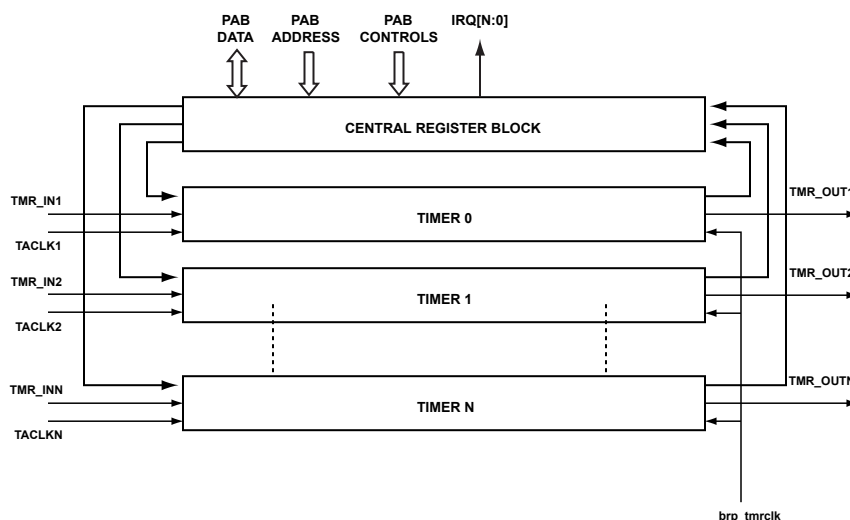


Figure 18-1: Timer Block Diagram

Internal Interface

The processor core always accesses the timer registers through the MMR access bus. Hardware ensures that all read and write operations from and to 32-bit timer registers are atomic. Every timer has a dedicated data interrupt request. There is also one common timer status and error interrupt request output that connects to the system event controller. Whenever a data interrupt request is generated, a data trigger master pulse is also driven out, if enabled. Each timer has an individual trigger input line, and each timer can be either started or stopped as a trigger slave.

In total, the GP timer module can have up to $(N + 1)$ interrupt request output lines and N data trigger lines.

Internal Timer Connections

The Timers support alternate inputs for the clock/capture (see [External Interface](#)). Some signals have internal default alternate connections according to the *Timer Signal Routing* table.

Table 18-5: Timer Signal Routing

Timer Signal	Connection
TM0_ACLK0	SYS_CLKIN0
TM0_ACI5	DAI0_CRS_PB04
TM0_ACLK5	DAI0_CRS_PB03
TM0_ACI6	DAI1_CRS_PB04
TM0_ACLK6	DAI1_CRS_PB03
TM0_ACI07	CNT0_TO signal
TM0_ACLK7	SYS_CLKIN0
TM0_ACI8	DAI0_PB06
TM0_ACLK8	DAI0_PB05

Table 18-5: Timer Signal Routing (Continued)

Timer Signal	Connection
TM0_ACI9	DAI1_PB06
TM0_ACLK9	DAI1_PB05

External Interface

Each GP timer module can support up to 16 individual timers. However, most processors have less than this number. The exact number of timers available on a given processor is available in the data sheet for the processor.

Every timer has one main input/output signal () and, usually, one auxiliary input pin, used as an alternate capture input (). Each timer can either run with a time base of SCLK or can reference an external clock on one of two pins. The TMR_ALT_CLK0 signal maps to individual alternate clock () pins for one or more timers. For instance, a TM_ACLK3 pin would provide an alternate site to supply an external signal that would serve as reference clock for TMR3. Likewise, the TMR_ALT_CLK1 signal from each timer unit connects together internally to provide a single global timer clock pin (TIMER_CLK) for the GP timer module. It is used as an additional time base.

GP Timer Operating Modes

The following sections provide information on the various operating modes of the GP timer.

General Operation

The core of every timer is a 32-bit counter that can be interrogated through the read-only `TIMER_TMR[n]_CNT` register. Once the module enables a timer, it loads the timer `TIMER_TMR[n]_CNT` register with a starting value.

A timer can operate in one of several different modes, configured through the `TIMER_TMR[n]_CFG` register for that timer. These modes are: PWMOUT, EXTCLK, WIDCAP, WATCHDOG, PININT, and IDLE. The *Timer Mode Descriptions* table summarizes the modes.

Table 18-6: Timer Mode Descriptions

Timer Mode	Description
PWMOUT	Generates single or continuous PWM waveforms with programmable pulse width, period, and delay
EXTCLK	Counts edges of an externally applied waveform
WIDCAP	Captures pulse width or period of an externally applied waveform
WATCHDOG	Monitors pulse width or period of an external signal and compares against a window of acceptable values, optionally generating an interrupt when it falls inside or outside of that window
PININT	Can generate an interrupt request on an active edge applied to a timer pin
IDLE	Idle; no activity

Period, Width and Delay Register Interaction

When the timer is started, writes to the buffer registers are immediately copied through to the double-buffered period, pulse width, and delay registers. These values are then ready for use in the first timer period. When a timer is already running, software can write new values to the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers. The written values are buffered and do not update into the registers until the end of the current period. (The update occurs when the value in the `TIMER_TMR[n]_CNT` register equals the value in the `TIMER_TMR[n]_PER` register.)

If new values are not written to these registers, the value from the previous period is reused. Writes to these registers are atomic; it is not possible for the high word to be written without the low word also being written. Values written to the period, pulse width, and delay registers are always stored in the buffer registers. Reads from the same register always return the current, active value of period, pulse width, or delay value. Written values are not readback until they become active.

The usage of the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers varies, depending on the mode of the timer specified by the `TIMER_TMR[n]_CFG.TMODE` bits. See the *Usage of the Period, Width, and Delay Registers in Different Timer Modes* table for more information.

Table 18-7: Usage of the Period, Width, and Delay Registers in Different Timer Modes

Timer Mode	<code>TIMER_TMR[n]_PER</code>	<code>TIMER_TMR[n]_WID</code>	<code>TIMER_TMR[n]_DLY</code>
IDLE	Not writable	Not writable	Not writable
WATCHDOG	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.	Read-only. Retains value of last measured width or period of the input signal.	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.
WIDCAP	Read-only. Period value captured at the appropriate time and updated from its buffer register simultaneously with the Width register.	Read-only. Width value captured at the appropriate time and updated from its buffer register simultaneously with the Period register.	Not used
PWMOUT	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.
EXTCLK	Can be updated on-the-fly.	Not used	Not used
PININT	Not used	Not used	Not used

If any of the period, pulse width, and delay registers are not used, then programs cannot write into that register. For example, in WIDCAP mode, the delay registers are not used. So, the program is not allowed to write any value to the `TIMER_TMR[n]_DLY` register. To prevent undesired operation, program the `TIMER_TMR[n]_CFG.TMODE` bits before programming the period, width, or delay registers.

If a program changes the `TIMER_TMR[n]_CFG.TMODE` bits from a status register to writable register (for example in PWMOUT mode), hardware does not clear these registers. These values are automatically overwritten by new values specified by software.

In PWMOUT mode with small periods, there may not be enough time between updates from the buffer registers to write these registers. The next period can use one old value and one new value. To prevent $(\text{width} + \text{pulse delay}) > \text{period}$ errors, write the width and delay registers before the period register when decreasing the values. Write the period register before the width and delay registers when increasing the value.

Single-Pulse PWMOUT Mode

In single-pulse PWMOUT mode, the timer generates a single pulse on the `TIMER_TMR[nn]` pin. This mode is frequently used to implement a precise delay, often with generating an output trigger. The timer module uses the value in the `TIMER_TMR[n]_DLY` register to control the assertion of a pulse. The value in the `TIMER_TMR[n]_WID` register defines the pulse width. The `TIMER_TMR[n]_PER` is not used and cannot be written in this mode. After completion of the pulse, the timer is automatically stopped, and optionally generates an interrupt. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to control pulse polarity.

The timer can be configured to generate a data interrupt request after satisfying various conditions specified by the `TIMER_TMR[n]_CFG.IRQMODE` bits.

It is not necessary to clear the relevant `TIMER_RUN` bit to stop the timer cleanly. At the end of the pulse, the timer stops automatically and the corresponding `TIMER_RUN` bit is cleared. To generate multiple discrete pulses (as opposed to a continuous PWM waveform), write a 1 to the appropriate `TIMER_RUN` bit, and wait for the timer to stop. Then, write another 1 to the same `TIMER_RUN` bit.

Continuous PWMOUT Mode

In continuous PWMOUT mode, the timer generates a repetitive pulse with a well-defined period, duty cycle, and pulse position. The `TIMER_TMR[n]_DLY`, `TIMER_TMR[n]_PER`, and `TIMER_TMR[n]_WID` registers are programmed with the values of the required PWM pulse. After the timer is started, the counter counts towards the value programmed in the `TIMER_TMR[n]_PER` register. Initially, the `TIMER_TMR[nn]` pin remains in a deasserted state. The pin toggles to an asserted state when the value in the `TIMER_TMR[n]_CNT` register equals the value in the `TIMER_TMR[n]_DLY` register.

The timer can control the assertion sense of the `TIMER_TMR[nn]` pin with the `TIMER_TMR[n]_CFG.PULSEHI` bit. The `TIMER_TMR[nn]` pin holds this value for the number of clock cycles specified in the `TIMER_TMR[n]_WID` register. Then, the pin deasserts and holds this value until the completion of the programmed period. The same waveform is generated repeatedly until the timer is disabled.

The timer can be configured to generate a data interrupt request after satisfying any of various conditions specified by the `TIMER_TMR[n]_CFG.IRQMODE` bits.

It is important to guarantee that the programmed period is greater than or equal to the sum of width and delay. Similarly, delay must be less than period. Violating either of these criteria results in an unpredictable waveform on the `TIMER_TMR[nn]` pin until the situation is rectified by writing proper values to these registers.

The maximum frequency possible to generate on the `TIMER_TMR[nn]` pin is achieved by setting `TIMER_TMR[n]_PER` to 2 and `TIMER_TMR[n]_WID` to 1. This operation makes the `TIMER_TMR[nn]` pin toggle each SCLK clock cycle (assuming the timer is configured to clock internally), producing a duty cycle of 50%.

When the `TIMER_STOP_CFG.TMR[nn]` bit of a timer is 0, the timer treats a stop operation as a stop-is-pending condition. When terminated with this setting, the timer automatically completes the current waveform and then stops cleanly, remaining in a deasserted state. This functionality prevents truncation of the current pulse and unwanted PWM patterns at the `TIMER_TMR[nn]` pin. The processor can determine when the timer stops running by polling the corresponding `TIMER_RUN.TMR[nn]` bit until it reads 0 or by waiting for the last interrupt (if enabled).

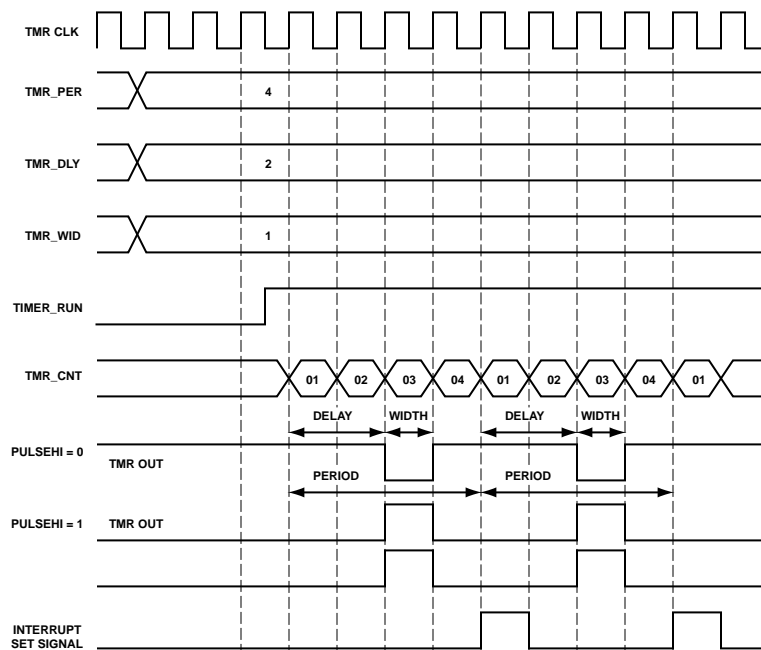


Figure 18-2: Signal Generation in Continuous PWMOUT Mode

The `TIMER_TMR[n]_CFG` register cannot be reconfigured until after the timer stops and the `TIMER_RUN` register reads 0.

Programs can force a timer to stop immediately in PWMOUT mode by writing a 1 to the `TIMER_STOP_CFG` register followed by writing a 1 to the `TIMER_RUN_CLR` register. (Or, a program can stop a timer by writing a 0 to the appropriate `TIMER_RUN.TMR[nn]` bit.) This operation stops the timer whether the pending stop is waiting for the end of the current period or the end of the current pulse width. The timer can use this feature to regain immediate control of a timer during an error recovery sequence.

Use this feature carefully, as it can corrupt the PWM pattern generated at the `TIMER_TMR[nn]` pin, though after such a stop the pin deasserts automatically. Each timer samples its `TIMER_RUN.TMR[nn]` bit at the end of each period. It stops cleanly at the end of the first period after the `TIMER_RUN.TMR[nn]` bit is low. A timer that is disabled and then restarted (before the end of the current period), continues to run as if nothing happened. Typically, the program disables a PWMOUT timer and then waits for it to stop itself.

Width Capture (WIDCAP) Mode

The timer uses WIDCAP mode, often called capture mode, to measure pulse widths on the `TIMER_TMR[nn]` or inputs. The polarity (active high or low) of the input signal can be selected with the `TIMER_TMR[n]_CFG.PULSEHI` bit. The *Timer Signal Flow in Width Capture Mode* figure shows the control signal flow for WIDCAP_CAP mode.

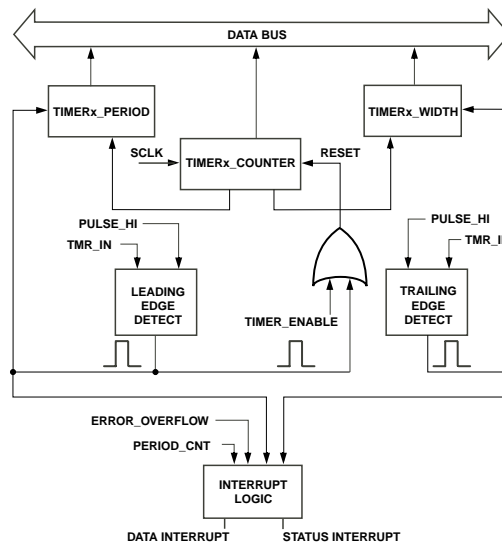


Figure 18-3: Timer Signal Flow in Width Capture Mode

NOTE: SCLK in the *Timer Signal Flow in Width Capture Mode* figure is SCLK.

In this mode, the timer uses the `TIMER_TMR[n]_CFG.TINSEL` bit to select between the `TIMER_TMR[nn]` or input. The internally clocked timer is used to determine the period and pulse width of the externally applied rectangular waveforms.

When a timer is enabled in this mode, the timer resets the count in its `TIMER_TMR[n]_CNT` register to 0x0000 0001. It does not start counting until it detects a leading edge on the selected input pin.

When the timer detects the first leading edge, it starts incrementing. When it detects a trailing edge of a waveform, it captures the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its width buffer register. At the next leading edge, the timer transfers the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its period buffer register. The `TIMER_TMR[n]_CNT` register is reset to 0x0000 0001 again, and the timer continues counting and capturing until it is disabled.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. The timer does not use the `TIMER_TMR[n]_DLY` register in this mode. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to control the definition of leading edge and trailing edge of the `TIMER_TMR[nn]` / pin.

In WIDCAP mode, the following events always occur at the same time as one unit:

1. The `TIMER_TMR[n]_PER` register is updated from the period buffer register.

2. The `TIMER_TMR[n]_WID` register is updated from the width buffer register.
3. The `TIMER_DATA_ILAT.TMR[nn]` bit is set (if enabled).
4. A timer data trigger pulse is generated (if enabled).

The `TIMER_TMR[n]_CFG.TMODE` bit 0 controls the point in time at which this set of events is executed. Taken together, these four events are called a measurement report. The `TIMER_STAT_ILAT` register is not set at a measurement report. A measurement report occurs, at most, once per input signal period. The current `TIMER_TMR[n]_CNT` value is always copied to the width buffer and period buffer registers at the trailing and leading edges of the input signal, respectively. But, these values are not visible to software. A measurement report event samples the captured values into visible registers and sets the timer interrupt request to signal that the `TIMER_TMR[n]_PER` and the `TIMER_TMR[n]_WID` registers are ready to be read.

When the `TIMER_TMR[n]_CFG.TMODE` bit = `b#1011`, the measurement report occurs just after the width buffer register captures its value at a falling edge. Then, the `TIMER_TMR[n]_WID` register reports the pulse width measured in the pulse that has ended, but the `TIMER_TMR[n]_PER` register reports the pulse period measured at the end of the previous period. If only the first trailing edge has occurred, then the first period value has not yet been measured at the first measurement report. So, the period value is not valid. A read of the `TIMER_TMR[n]_PER` value in this case returns 0. See the *Example of Width Capture Deasserted Mode (TMODE=b#1011)* figure for more information.

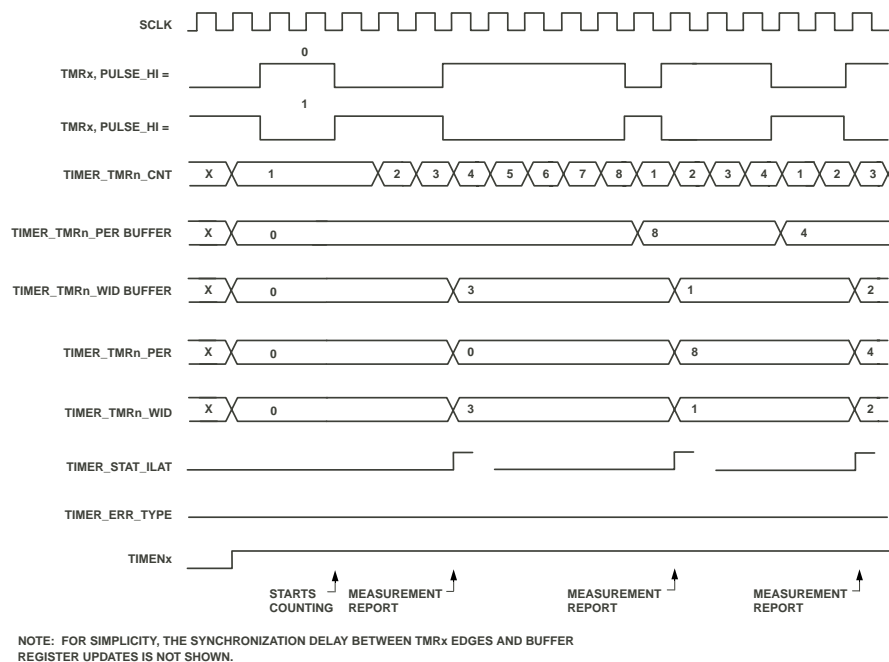


Figure 18-4: Example of Width Capture Deasserted Mode (TMODE=b#1011)

NOTE: SCLK in the *Example of Width Capture Deasserted Mode (TMODE=b#1011)* figure is SCLK.

When the `TIMER_TMR[n]_CFG.TMODE` bit = `b#1010`, the measurement report occurs just after the period buffer register captures its value at a leading edge. Then, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID`

registers report the pulse period and pulse width measured in the period that has ended. Refer to the *Example of Width Capture Asserted Mode (TMODE=b#1010)* figure for more information.

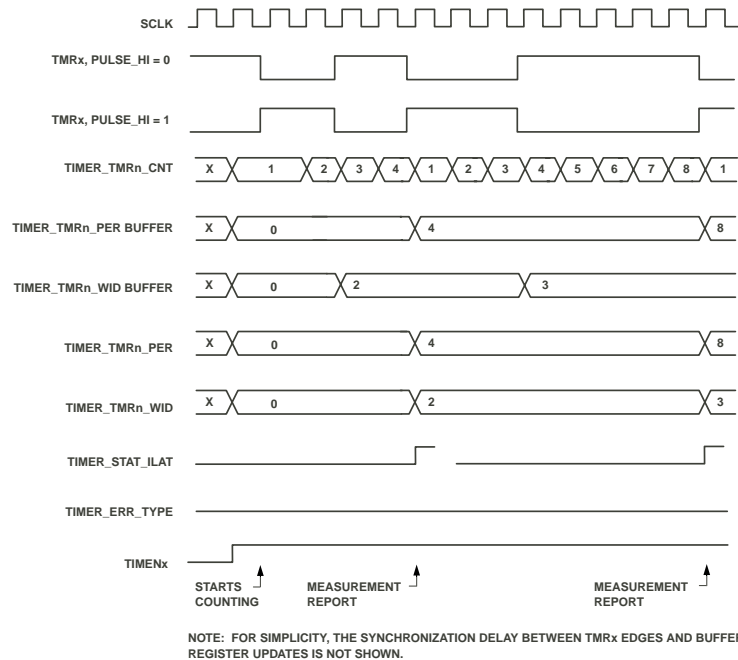


Figure 18-5: Example of Width Capture Asserted Mode (TMODE=b#1010)

NOTE: SCLK in the *Example of Width Capture Asserted Mode (TMODE=b#1010)* figure is SCLK.

To measure the pulse width of a waveform that has only one leading edge and one trailing edge, set `TIMER_TMR[n]_CFG.TMODE = b#1011`. If `TIMER_TMR[n]_CFG.TMODE = b#1010` for this case, no period value is captured in the period buffer register. Instead, the timer generates an error report interrupt request (if enabled) when the `TIMER_TMR[n]_CNT` range is exceeded and the counter wraps around. In this case, both the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers read 0 (because no measurement report occurred to copy the value captured in the width buffer register to the `TIMER_TMR[n]_WID` register).

If using the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010 mode to measure the width of a single pulse, programs can disable the timer after taking the interrupt that ends the measurement interval. If desired, restart the timer as appropriate in preparation for another measurement. This procedure prevents the timer from free-running after the width measurement and logging errors generated by the timer count overflowing.

Width Capture Mode Overflow

A timer status interrupt request (when enabled) is generated when the `TIMER_TMR[n]_CNT` register wraps around from 0xFFFF FFFF to 0 in the absence of a leading edge. At that point, the `TIMER_STAT_ILAT` bit is set and the `TIMER_ERR_TYPE` bits change to indicate a count overflow due to a period greater than the range of the counter. This indication is referred to as an error report. A data interrupt request in WIDCAP mode indicates that a new measurement is ready to be read (a measurement report). Similarly, an interrupt request on the timer status interrupt line (shared interrupt request for all timers) indicates an overflow error when generated in this mode.

The `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are never updated at the time an overflow error is signaled. If the timer overflows and the `TIMER_TMR[n]_CFG.TMODE` bit = `b#1010`, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are not updated. If the timer overflows and the `TIMER_TMR[n]_CFG.TMODE` bit = `b#1011`, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are updated only if a trailing edge is detected at a previous measurement report.

Software can count the number of error reports between measurement report interrupt requests to measure input signal periods longer than `0xFFFF FFFF`. Each error report interrupt request adds a full 2^{32} SCLK counts to the total for the period, but the width is ambiguous. Ensure that if software monitors only the status interrupt request, then status interrupt requests from all other timers are masked.

Refer to the *Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)* figure. The period is `0x1 0000 0004`, but the pulse width could be either `0x0 0000 0002` or `0x1 0000 0002`.

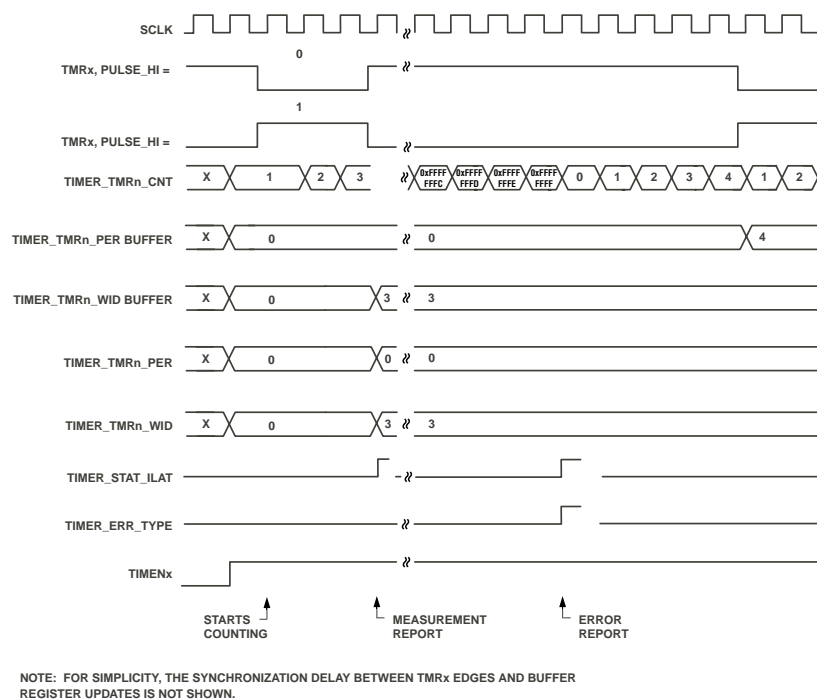


Figure 18-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

NOTE: SCLK in the *Example Timing for Width Capture Followed by Period Overflow* figure is SCLK.

The waveform applied to the `TIMER_TMR[nn]` (or) pin is not required to have a 50% duty cycle. The minimum input low time is little more than one SCLK period. The minimum input high time is a little more than one SCLK period. (Refer to the product data sheet for details.) The maximum `TIMER_TMR[nn]` input frequency is less than $SCLK/2$, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer measures: period = 2 and pulse width = 1.

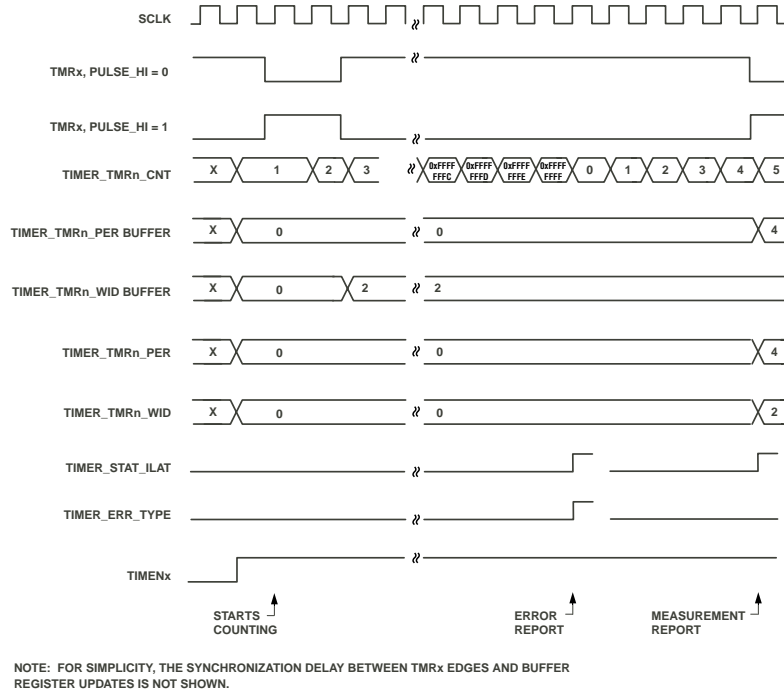


Figure 18-7: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

NOTE: SCLK in the *Example Timing for Width Capture Followed by Period Overflow* figure is SCLK.

Windowed Watchdog (WATCHDOG) Modes

In windowed watchdog (WATCHDOG) modes, a timer can take inputs from either the pin or the pin. With this mode, the timer can monitor pulse width (width watchdog mode) or pulse period (period watchdog mode) on the input line. It also compares the measured value against a minimum required value and maximum allowed value and generates an interrupt request appropriately. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to select polarity of the input signal.

The waveform applied to the input pin in watchdog mode is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet.

Windowed Watchdog Width Mode

- Inside the window ($p_{MIN} < \text{pulse width} \leq p_{MAX}$), or
- Outside the window ($\text{pulse width} \leq p_{MIN}$ or $\text{pulse width} > p_{MAX}$)

After enabling the timer in this mode, it always starts counting at the asserting edge of the input signal. Any pulse that is already active when the timer is enabled is ignored.

With the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11, the timer generates an interrupt if the timed pulse width exceeds p_{MAX} , or if the pulse width is less than p_{MIN} . After attaining p_{MAX} , the pulse stays at an active level, and the counter keeps on counting until it sees a deasserting edge. When the input pulse is not active, the counter holds its

current value until it again sees an asserting edge, or it restarts. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE = b#10`.

In this mode, a trailing edge on the input pin triggers capturing of pulse width into the `TIMER_TMR[n]_WID` register. During the inactive portion of the input signal, the internal counter does not increment. The *Watchdog Width Mode Timing* figure shows the signal flow in this mode.

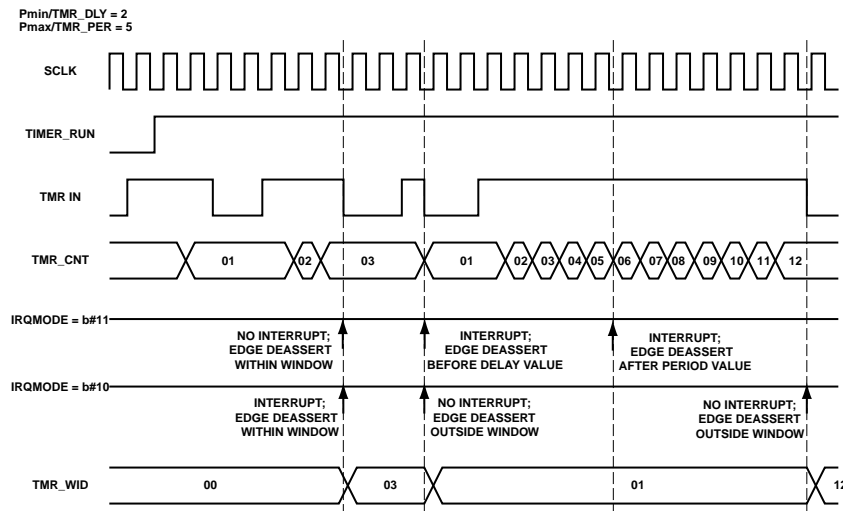


Figure 18-8: Watchdog Width Mode Timing

NOTE: SCLK in the *Watchdog Width Mode Timing* figure is SCLK.

To check only the upper limit on pulse width (p_{MAX} but not p_{MIN}), then program p_{MIN} as 0 or 1. In such a case, it is better to use `TIMER_TMR[n]_CFG.IRQMODE = b#11`. With `TIMER_TMR[n]_CFG.IRQMODE = b#10`, a pulse width of 1 clock cycle results in an interrupt. For details, see the *Windowed Watchdog Width Mode Interpretation* table.

Table 18-8: Windowed Watchdog Width Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
0 or 1	Anything ≥ 1	PW = 1	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW \leq TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt

Table 18-8: Windowed Watchdog Width Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
> 1 but \leq (Period -1)	Anything > 1	$PW \leq TMR_DLY$	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		$TMR_DLY < PW \leq TMR_PER$	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		$PW > TMR_PER$	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
\geq Period	-	$PW \leq TMR_PER$	Undefined	Undefined	No Error Interrupt
	-	$PW > TMR_PER$	Undefined	Undefined	b#11 Error Type
-	0	-	Undefined	Undefined	b#10 Error Type

Windowed Watchdog Period Mode

In this mode, the timer monitors the number of clock cycles between two consecutive rising or falling edges of an input signal on either the or pin. Program the required minimum number of clock cycles (t_{MIN}) in the `TIMER_TMR[n]_DLY` register and the required maximum allowed number of clock cycles (t_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SCLK) or alternate time clock (). The timer can generate an interrupt when two consecutive occurrences of an active edge are:

- Within a specified window ($t_{MIN} < \text{Pulse Period} \leq t_{MAX}$), or
- Outside a specified window ($\text{pulse width} \leq (t_{MIN} \text{ or } t_{MAX} < \text{pulse width})$)

When the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11 and the pulse period $> t_{MAX}$ or is $\leq t_{MIN}$, the timer generates an interrupt (if unmasked). After attaining the t_{MAX} value, the counter keeps on counting until it sees an active edge on the input line. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE` =b#10. Refer to the *Watchdog Period Mode Timing* figure for timer functionality in period watchdog mode.

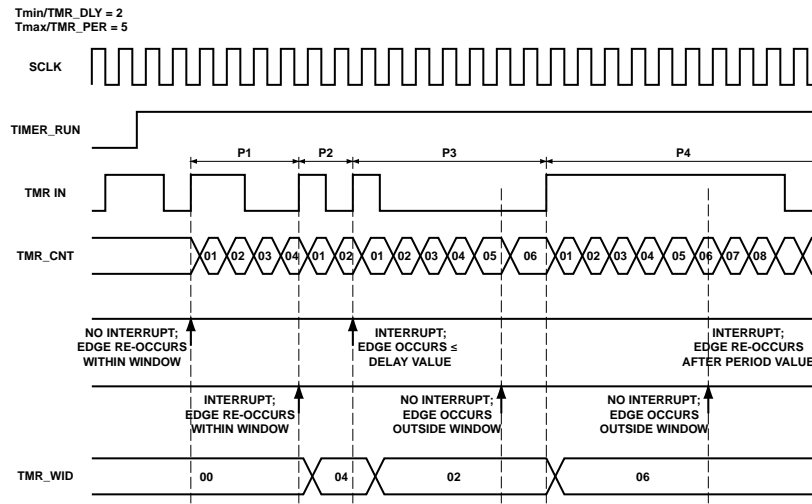


Figure 18-9: Watchdog Period Mode Timing

NOTE: SCLK in the *Watchdog Period Mode Timing* figure is SCLK.

To check only the upper limit on period (the t_{MAX} value, not the t_{MIN} value), program t_{MIN} as 0 or 1. For details, refer to the *Windowed Watchdog Period Mode Interpretation* table.

Table 18-9: Windowed Watchdog Period Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
0 or 1	Anything ≥ 2	Pulse Period \leq TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period $>$ TMR_PER	No Interrupt	Interrupt when pulse period crosses Pmax (Period Register) value	No Error Interrupt
> 1 but \leq Period -1	Anything ≥ 2	Pulse Period \leq TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY $<$ Pulse Period \leq TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period $>$ TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) value	No Error Interrupt
\geq Period	-	Pulse Period $<$ TMR_PER	Undefined	Undefined	No Error Interrupt

Table 18-9: Windowed Watchdog Period Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
		Pulse Period \geq TMR_PER	Undefined	Undefined	b#11 Error Type
-	0 or 1	-	Undefined	Undefined	b#10 Error Type

Pin Interrupt (PININT) Mode

In PININT mode, any active edges on either the `TIMER_TMR[nn]` pin or the pin can cause an edge-based interrupt, if enabled. (The timer uses the `TIMER_TMR[n]_CFG.TINSEL` register to select the pin). The event on the input pin can set the `TIMER_DATA_ILAT.TMR[nn]` bit and issue a system interrupt request. Program the `TIMER_TMR[n]_CFG.PULSEHI` bit to change active edge polarity.

Since the interrupt request is generated in the SCLK clock domain, the width of the input signal must be more than one SCLK period. Along with generating the interrupt request, the timer also generates a trigger pulse (configured using the `TIMER_TRG_MSK` register). Due to the configuration of polarity, glitches can cause the generation of an undesired interrupt request at the input. To avoid this problem, programs must ensure that interrupt requests are unmasked only after configuring the desired polarity.

External Clock (EXTCLK) Mode

The timer uses EXTCLK mode, sometimes referred to as the counter mode, to count external events (signal edges), on either the `TIMER_TMR[nn]` or input pin. The timer works as a counter clocked by an external source (the signal at the pin), which can be asynchronous to SCLK. The current count in the `TIMER_TMR[n]_CNT` register represents the number of leading-edge events detected. The `TIMER_TMR[n]_PER` register is programmed with the value of the maximum timer external count before stopping or issuing an interrupt request or trigger.

The `TIMER_TMR[n]_CFG.PULSEHI` bit determines the polarity of the leading edge on the input pin. The timer uses the `TIMER_TMR[n]_CFG.TINSEL` bit to select whether the event is counted on the `TIMER_TMR[nn]` or on the pin. The `TIMER_STAT_ILAT.TMR[nn]` and `TIMER_ERR_TYPE` bits are set if *one* of these conditions is met:

- `TIMER_TMR[n]_CNT` wraps around from 0xFFFF FFFF to 0
- The period = 0 at startup
- `TIMER_TMR[n]_CNT` register rolls over (from count = period to count = 0x1)

The `TIMER_TMR[n]_WID` and `TIMER_TMR[n]_DLY` registers are unused in this mode and must not be written.

The *EXTCLK Mode Control Flow* figure shows a flow diagram for EXTCLK mode.

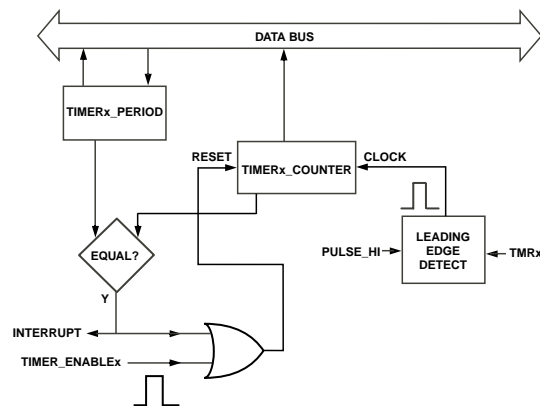


Figure 18-10: EXTCLK Mode Control Flow

The waveform applied to the input pin is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet. Program the period to any value from 1 to $(2^{32} - 1)$, inclusive.

After the timer has started, it resets the `TIMER_TMR[n]_CNT` register to 0x0 and then waits for the first leading edge on the input pin. This edge causes `TIMER_TMR[n]_CNT` to be incremented to the value 0x1, and every subsequent leading edge increments it by one. After the `TIMER_TMR[n]_CNT` register reaches the value programmed in the `TIMER_TMR[n]_PER` register, the corresponding `TIMER_DATA_ILAT` bit is set, and an interrupt and trigger are both generated (if enabled). The next leading-edge reloads the `TIMER_TMR[n]_CNT` register with 0x1, and the timer continues counting until it is disabled.

GP Timer Programming Concepts

Using the features, operating modes, and event control for the GP timer to their greatest potential requires an understanding of some GP timer-related concepts.

Setting Up Constantly Changing Timer Conditions

This task shows how to use different period, pulse width, and delay settings for each of the first three timer periods after the timer starts.

1. Program the first set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.
2. Enable the timer using the `TIMER_RUN` register.
3. Immediately program the second set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values, as needed.
4. Wait for the first timer interrupt request.
5. Program the third set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.

Each new setting is then programmed when the preceding timer interrupt request is received.

Configuring, Enabling, and Disabling One or More Timers

1. Configure the relevant timers for the operating mode and other properties using the `TIMER_TMR[n]_CFG` register.
2. Write a 1 to the representative `TIMER_RUN.TMR[nn]` bit. Or, use the `TIMER_RUN_SET` register to avoid disturbing the settings of other timers that are not going through configuration.

The timer is enabled and operating.

3. To stop one or more timers, first program the `TIMER_STOP_CFG` register to determine whether to stop immediately or gracefully upon receiving a stop command.

ADDITIONAL INFORMATION: PWMOUT modes are the only modes where a timer can be configured for graceful termination.

4. Write a 0 to the representative `TIMER_RUN.TMR[nn]` bits to stop the timer according to their `TIMER_STOP_CFG` settings. Alternately, write a 1 to the appropriate `TIMER_RUN_CLR.TMR[nn]` bits to avoid disturbing the settings of other timers that are not terminating.

The timers stop.

Configuring Timer Data and Status Interrupts

1. Configure the `TIMER_TMR[n]_CFG.IRQMODE` bit field with the desired interrupt properties.
2. Unmask the interrupt source at the system event controller.
3. Set the `TIMER_TMR[n]_CFG.IRQMODE` field but leave the interrupt masked at the system level to poll the `TIMER_DATA_ILAT.TMR[nn]` bit of the timer without generating an interrupt.
4. Use the `TIMER_STAT_IMSK` register to generate interrupt requests by overflow or error conditions (incorrect programming values). The timer uses the `TIMER_STAT_ILAT.TMR[nn]` bits to report interrupt errors, when the timer status interrupt source is unmasked at the system event controller.
5. To poll the `TIMER_STAT_ILAT.TMR[nn]` bit of the timer without generating an interrupt, unmask the corresponding bit in the `TIMER_STAT_IMSK` register, but leave the interrupt masked at the system level.

Configuring the Timer as a Trigger Slave

The timer can be configured to either start or stop or toggle between these two states on the input trigger pulse depending on the configuration of the `TIMER_TMR[n]_CFG.TGLTRIG` and `TIMER_TMR[n]_CFG.SLAVETRIG` bits.

- If `TIMER_TMR[n]_CFG.TGLTRIG` bit =0 and `TIMER_TMR[n]_CFG.SLAVETRIG` bit =1 then the trigger pulse starts timer, if it is stopped.

- If `TIMER_TMR[n]_CFG.TGLTRIG` bit =0 and `TIMER_TMR[n]_CFG.SLAVETRIG` bit =0 then the trigger pulse stops timer, if it is running.
- When `TIMER_TMR[n]_CFG.TGLTRIG` bit =0, the trigger pulse has no effect when the timer is already in the requested state.

If `TIMER_TMR[n]_CFG.TGLTRIG` bit is 1, the trigger pulse starts the timer if it is stopped, or stops the timer if it is running. The `TIMER_TMR[n]_CFG.SLAVETRIG` bit has no effect on trigger mechanism. In continuous PWMOUT mode, the timer stops gracefully or abruptly depending on the stop mechanism programmed in the `TIMER_STOP_CFG_CLR` register. In other modes, the timer stops immediately.

Ordered Trigger Toggle Mode

The timer can be configured to either start or stop the timer on the input trigger pulse depending on the configuration of the `TIMER_TMR[n]_CFG.TGLTRIG` and `TIMER_TMR[n]_CFG.ORDTGLTRIG` bits.

- If `TIMER_TMR[n]_CFG.TGLTRIG` bit is 1 and `TIMER_TMR[n]_CFG.ORDTGLTRIG` bit is 0, then the input trigger pulse on trigger slave 0 or trigger slave 1 starts the timer if it is stopped, or stops the timer if it is running.
- If `TIMER_TMR[n]_CFG.TGLTRIG` bit is 1 and `TIMER_TMR[n]_CFG.ORDTGLTRIG` bit is 1, then the input trigger pulse on trigger slave 0 starts timer if it is stopped. If the timer is running, it stops when the input trigger pulse on trigger slave 1 is detected.

Trigger slave 0 cannot toggle the timer from the run to halt state. Trigger slave 1 cannot toggle the timer from the halt to run state.

NOTE: The timer state is not toggled by writes to the `TIMER_RUN`, `TIMER_RUN_SET`, and `TIMER_RUN_CLR` registers in this mode.

Using the Timer Broadcast Feature

The broadcast feature provides a means to update period, width, and delay registers simultaneously across more than one timer.

1. Enable the appropriate broadcast bits (`TIMER_TMR[n]_CFG.BPEREN`, `TIMER_TMR[n]_CFG.BWIDEN` are `TIMER_TMR[n]_CFG.BDLYEN`) for the timers involved in the broadcast. The use of these bits depends on which broadcast registers the timer uses (`TIMER_BCAST_PER`, `TIMER_BCAST_WID`, or `TIMER_BCAST_DLY`).
2. Program the `TIMER_BCAST_PER` register (for example), to broadcast the period setting across the multiple timers enabled.

The enabled timers load their `TIMER_TMR[n]_PER` registers with the value specified in the `TIMER_BCAST_PER` register.

3. Repeat Step 2 as needed for the `TIMER_BCAST_WID` and `TIMER_BCAST_DLY` register settings.

Timer Illegal States

The following sections use these definitions:

- Startup. The first clock period during which the timer counter is running after the timer is started by writing the `TIMER_RUN` register.
- Rollover. The time when the current count in `TIMER_TMR[n]_CNT` matches the value in `TIMER_TMR[n]_PER` and the counter is reloaded with the value 1.
- Overflow. The timer counter was incremented instead of doing a rollover when it was holding the maximum count value of `0xFFFF FFFF`. The counter does not have a large enough range to express the next greater value and so it erroneously loads a new value of `0x0000 0000`.
- Unchanged. No new error.

When the `TIMER_ERR_TYPE` register is designated unchanged, it displays the previously reported error code orb# 00 when there has been no error since this timer was enabled.

When the `TIMER_STAT_ILAT` register is unchanged, it reads 0 when there has been no error or overflow since this timer was enabled. Or, it reads 0 if software has performed a W1C to clear any previous error. If software has not acknowledged a previous error, the `TIMER_STAT_ILAT` register reads 1. Software can read the `TIMER_STAT_ILAT` register to check for errors. If a particular bit of a timer is set in this register, software can then read the `TIMER_ERR_TYPE` register for more information. Once detected, software can W1C the appropriate `TIMER_STAT_ILAT` bit to acknowledge the error.

Read the following tables as:

- In mode ___ at event __,
- if `TIMER_TMR[n]_PER` is ___ and `TIMER_TMR[n]_WID` is ___ and `TIMER_TMR[n]_DLY` is __,
- then `TIMER_ERR_TYPE` is ___ and `TIMER_STAT_ILAT` is ___.

Startup error conditions do not prevent the timer from starting. Similarly, overflow and rollover error conditions do not stop the timer. Illegal cases can cause unwanted behavior of the pin.

NOTE: For PININT mode, the timer does not use error functionality.

Continuous PWMOUT Mode

Table 18-10: Startup Event

<code>TIMER_TMR[n]_PER</code>	<code>TIMER_TMR[n]_DLY</code>	<code>TIMER_TMR[n]_WID</code>	<code>TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY</code>	<code>TIMER_ERR_TYPE</code>	<code>TIMER_STAT_ILAT</code> (if enabled)
≤ 1	Anything other than period[8]	Anything	Anything	b#10	Set

Table 18-10: Startup Event (Continued)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	\leq PERIOD	Unchanged	Unchanged
	Anything including 0	Anything including 0	$>$ PERIOD	Unchanged[9] (Detected at rollover)	Unchanged (Detected at rollover)
	Anything	Anything	$> 2^{32} - 1$	b#11	Set
	=Period	=0	=Period	No error	Unchanged (Detected at rollover)

Table 18-11: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≥ 1	Anything	Anything	Anything	b#10[timer running at SCLK] b#11 [timer running at ALT_CLKx]	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	\leq PERIOD	Unchanged	Unchanged
	Anything including 0, excluding TMR_PER value	Anything >0	$>$ PERIOD	b#11	Set
	Anything	Anything	$> 2^{32} - 1$	b#11	Set
	= Period[10]	=0	=Period	b#11	Set
	$>$ Period	=0	$>$ Period	Unchanged	Unchanged

Table 18-12: Overflow Event (On TMR_PER Register Programming Error Only)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

Single Pulse PWMOUT Mode

For single pulse PWMOUT mode, there are no rollover events.

Table 18-13: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_STAT_ILAT (if enabled)	TIMER_STAT_ILAT (if enabled)
N/A	Anything	== 0	Anything	b#11[11]	Set
N/A	Anything including 0	≥1	> 2 ³² - 1	Unchanged	Unchanged
N/A	Anything including 0	≥1	> 2 ³² - 1	b#11	Set

Table 18-14: Overflow Event (On another error, such as DELAY + WIDTH ≥ 2³²- 1)

	TIMER_TMR[n]_DLY		TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY		TIMER_STAT_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

WIDCAP Mode

For WIDCAP mode, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are read-only and the `TIMER_TMR[n]_DLY` register is not used. Therefore, no startup or rollover errors are possible.

Table 18-15: Overflow Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	Anything	N/A	b#01	Set

EXTCLK Mode

Table 18-16: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 18-17: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 18-18: Overflow Event (On TMR_PER Register = 0 Only)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	N/A	N/A	b#01	Set

WATCHDOG Events

Table 18-19: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[12]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything < PERIOD	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 18-20: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[10]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 18-21: Overflow Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	N/A	N/A	b#01	Set

ADSP-2156x TIMER Register Descriptions

General-Purpose Timer Block (TIMER) contains the following registers.

Table 18-22: ADSP-2156x TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

Broadcast Delay Register

For timers with `TIMER_TMR[n]_CFG.BDLYEN` enabled, a write to the `TIMER_BCAST_DLY` register concurrently updates the delay (`TIMER_TMR[n]_DLY`) registers of only those timers. A read of the `TIMER_BCAST_DLY` register returns `0x00000000`, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_DLY` register.

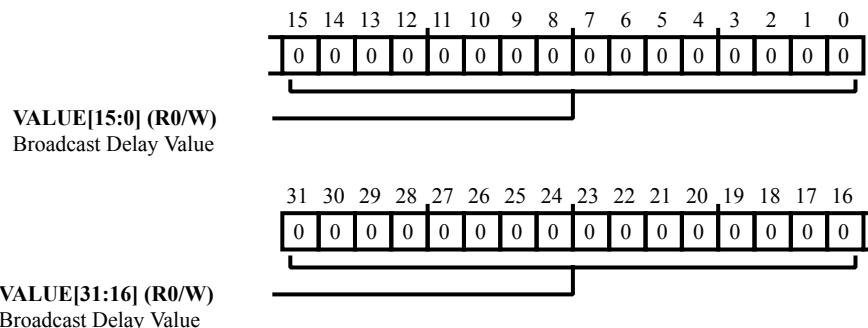


Figure 18-11: `TIMER_BCAST_DLY` Register Diagram

Table 18-23: `TIMER_BCAST_DLY` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Delay Value. A write to the <code>TIMER_BCAST_DLY.VALUE</code> bit field concurrently updates the delay (<code>TIMER_TMR[n]_DLY</code>) registers of only those timers. A read of the <code>TIMER_BCAST_DLY.VALUE</code> bit field returns <code>0x0000 0000</code> , and no bus error is generated.

Broadcast Period Register

For timers with `TIMER_TMR[n]_CFG.BPEREN` enabled, a write to the `TIMER_BCAST_PER` register concurrently updates the period (`TIMER_TMR[n]_PER`) registers of only those timers. A read of `TIMER_BCAST_PER` returns `0x00000000`, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_PER` register.

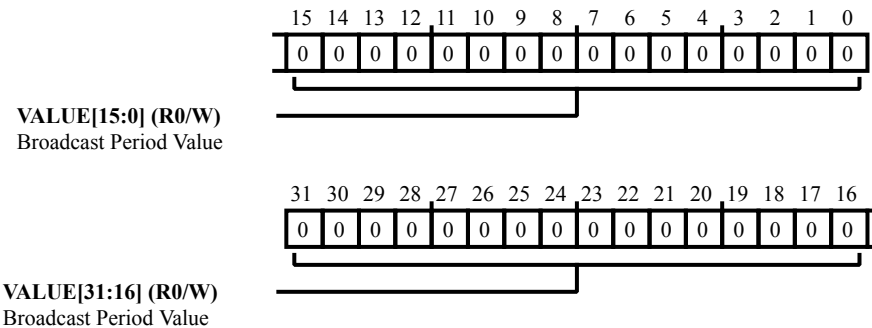


Figure 18-12: `TIMER_BCAST_PER` Register Diagram

Table 18-24: `TIMER_BCAST_PER` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Period Value. A write to the <code>TIMER_BCAST_PER.VALUE</code> bit field concurrently updates the period (<code>TIMER_TMR[n]_PER</code>) registers of only those timers. A read of the <code>TIMER_BCAST_PER.VALUE</code> bit fields returns <code>0x0000 0000</code> , and no bus error is generated.

Broadcast Width Register

For timers with `TIMER_TMR[n]_CFG.BWIDEN` enabled, a write to the `TIMER_BCAST_WID` register concurrently updates the width (`TIMER_TMR[n]_WID`) registers of only those timers. A read of the `TIMER_BCAST_WID` register returns `0x00000000`, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_WID` register.

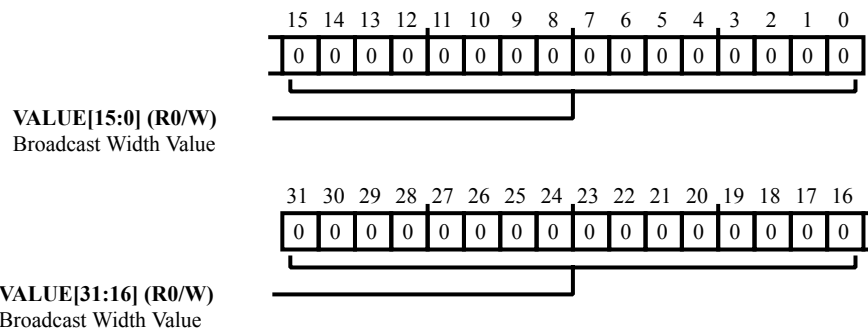


Figure 18-13: `TIMER_BCAST_WID` Register Diagram

Table 18-25: `TIMER_BCAST_WID` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Width Value. A write to the <code>TIMER_BCAST_WID.VALUE</code> bit field concurrently updates the width (<code>TIMER_TMR[n]_WID</code>) registers of only those timers. A read of the <code>TIMER_BCAST_WID.VALUE</code> bit field returns <code>0x0000 0000</code> , and no bus error is generated.

Data Interrupt Latch Register

The `TIMER_DATA_ILAT` holds the latched interrupt status for interrupt requests that have been unmasked (enabled) by the `TIMER_DATA_IMSK` register and generated according to the conditions selected by the `TIMER_TMR[n]_CFG.IRQMODE` bits. If a bit in `TIMER_DATA_ILAT` is already set and the corresponding interrupt is masked in `TIMER_DATA_IMSK`, the latch holds its old value, leaving the interrupt request asserted until it is reset by software with a `W1C` operation.

Note that interrupt service routines (ISRs) should clear the appropriate bits in `TIMER_DATA_ILAT` before returning from the ISR, to ensure that the interrupt is not re-issued. To make sure that no timer event is missed, the latch should be reset at the very beginning of the ISR when in `EXTCLK` mode.

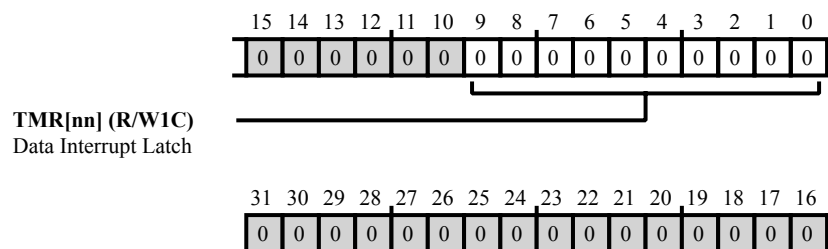


Figure 18-14: `TIMER_DATA_ILAT` Register Diagram

Table 18-26: `TIMER_DATA_ILAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W1C)	<code>TMR[nn]</code>	Data Interrupt Latch. For all <code>TIMER_DATA_ILAT.TMR[nn]</code> bits, status of =0 indicates no interrupt is latched, and status of =1 indicates a latched interrupt (indicating an unmasked interrupt request from a timer with a condition matching the one selected with corresponding <code>TIMER_TMR[n]_CFG.IRQMODE</code> bit has occurred).

Data Interrupt Mask Register

Each timer may generate a unique processor data interrupt request signal. The `TIMER_DATA_IMSK` register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the `TIMER_DATA_IMSK` register is 0xFFFF, masking these interrupts after reset.

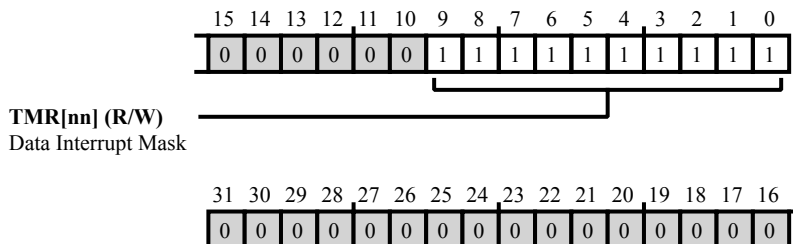


Figure 18-15: `TIMER_DATA_IMSK` Register Diagram

Table 18-27: `TIMER_DATA_IMSK` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	TMR[nn]	Data Interrupt Mask. For all <code>TIMER_DATA_IMSK.TMR[nn]</code> bits, write =0 unmask (enables) the corresponding data interrupt request, and write =1 masks (disables) the corresponding data interrupt request.

Error Type Status Register

The `TIMER_ERR_TYPE` register contains error type status bits for each timer. These bits indicate the type of error (if any) in a running timer. This register is read-only. These status bits are cleared at reset and when a particular timer is enabled.

Each time an error request interrupt is latched in the `TIMER_STAT_ILAT` register, the corresponding `TERRx` bits in the `TIMER_ERR_TYPE` register are loaded with a code that identifies the type of error that was detected. This status value is held until the next error or until a particular timer is restarted. No bus error is generated if a write is performed on the `TIMER_ERR_TYPE` register.

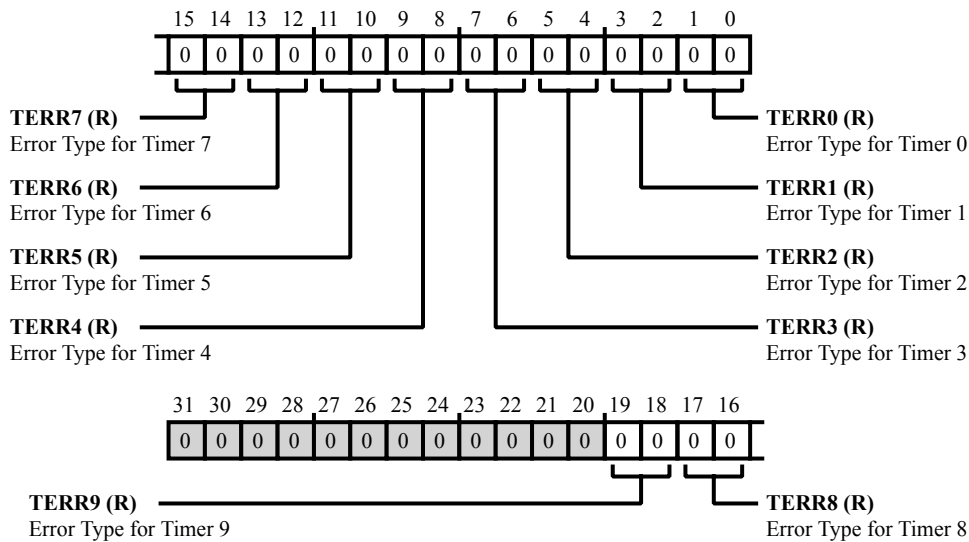


Figure 18-16: `TIMER_ERR_TYPE` Register Diagram

Table 18-28: `TIMER_ERR_TYPE` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:18 (R/NW)	TERR9	Error Type for Timer 9.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
17:16 (R/NW)	TERR8	Error Type for Timer 8.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Table 18-28: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:14 (R/NW)	TERR7	Error Type for Timer 7.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
13:12 (R/NW)	TERR6	Error Type for Timer 6.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
11:10 (R/NW)	TERR5	Error Type for Timer 5.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
9:8 (R/NW)	TERR4	Error Type for Timer 4.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
7:6 (R/NW)	TERR3	Error Type for Timer 3.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
5:4 (R/NW)	TERR2	Error Type for Timer 2.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error

Table 18-28: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:2 (R/NW)	TERR1	Error Type for Timer 1.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error
1:0 (R/NW)	TERR0	Error Type for Timer 0.
		0 No Error
		1 Counter Overflow Error
		2 PER Register Programming Error
		3 WID or DLY Register Programming Error

Run Register

The `TIMER_RUN` allows all timers to be enabled simultaneously, permitting them to run synchronously. For each timer, there is a single start/stop control bit. Writing a 1 to this bit starts the corresponding timer; writing a 0 stops the timer with mechanism specified in the timer stop configuration `TIMER_STOP_CFG` register.

The start/stop control bits can be set/reset individually or in any combination. While starting or stopping one particular timer directly with this register, software must perform a read-modify write, so the bits corresponding to other timers remain unchanged. To avoid this need, software can use the `TIMER_RUN_CLR` register.

Reading the `TIMER_RUN` register shows the start status for the corresponding timer. A 1 indicates that the timer is running.

If a timer is in run state (corresponding run bit is =1), a software write of 1 in this bit does not have any effect on the timer state. The write does not result in restarting the timer.

Note that the `TIMER_RUN` register is not used in PININT mode. PININT mode starts as soon as the `TIMER_TMR[n]_CFG.TMODE` bits are set to 111.

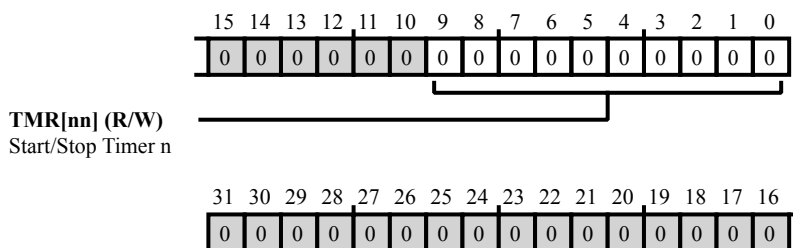


Figure 18-17: TIMER_RUN Register Diagram

Table 18-29: TIMER_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	TMR[nn]	Start/Stop Timer n. For all <code>TIMER_RUN.TMR[nn]</code> bits, write =0 for stop, and write =1 for start. Read =1 when timer is running.

Run Clear Register

The `TIMER_RUN_CLR` register is an alias register, providing a mechanism to clear a specific start/stop bit in the `TIMER_RUN` register without affecting other bits in `TIMER_RUN`. To stop a particular timer, software must write a 1 into the corresponding `TIMER_RUN_CLR` bit. Writing a 0 has no effect. Because `TIMER_RUN_CLR` is a write-only register, the result of any write to this register must be checked by reading the `TIMER_RUN` register. A read of the `TIMER_RUN_CLR` returns 0x0000.

Note that the stopping mechanism of a timer may be abrupt or graceful (after completion of current waveform period) depending on the selection in the `TIMER_STOP_CFG` register.

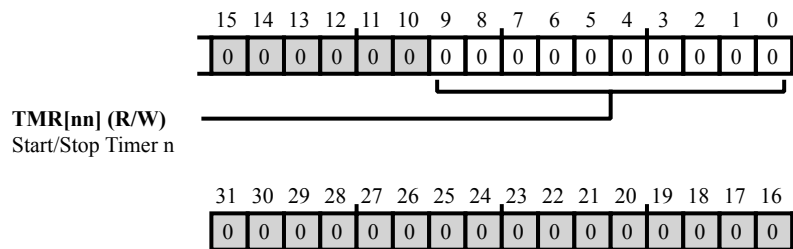


Figure 18-18: `TIMER_RUN_CLR` Register Diagram

Table 18-30: `TIMER_RUN_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R0/W1C)	<code>TMR[nn]</code>	RUN Clear Alias. For all <code>TIMER_RUN_CLR.TMR[nn]</code> bits, write =0 has no effect, and write =1 for stop (clearing the corresponding in start/stop bit in the <code>TIMER_RUN</code> register). Using <code>TIMER_RUN_CLR</code> to clear start/stop bits permits stopping specific timers without influencing run status of other timers.

Run Set Register

The `TIMER_RUN_SET` register is an alias register, providing a mechanism to set a specific start/stop bit in the `TIMER_RUN` register without affecting other bits in `TIMER_RUN`. To start a particular timer, software must write a 1 into the corresponding `TIMER_RUN_SET` bit. Writing a zero has no effect. For an example, to start timer 3 without affecting any other timer, write 0x0008 into `TIMER_RUN_SET`. Because `TIMER_RUN_SET` is a write-only register, the result of any write to this register must be checked by reading the `TIMER_RUN` register. A read of the `TIMER_RUN_SET` returns 0x0000.

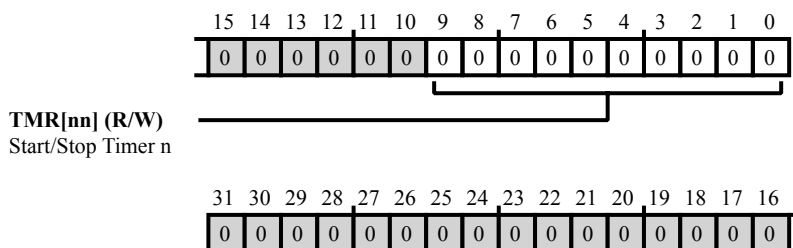


Figure 18-19: `TIMER_RUN_SET` Register Diagram

Table 18-31: `TIMER_RUN_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R0/W1S)	<code>TMR[nn]</code>	RUN Set Alias. For all <code>TIMER_RUN_SET.TMR[nn]</code> bits, write =0 has no effect, and write =1 for start (setting the corresponding start/stop bit in the <code>TIMER_RUN</code> register). Using <code>TIMER_RUN_SET</code> to set start/stop bits permits starting specific timers without influencing the run status of other timers.

Status Interrupt Latch Register

The `TIMER_STAT_ILAT` holds the latched interrupt status for error interrupt requests, indicating a timer overflow condition or indicating that prohibited programming has occurred for a timer. These interrupt status bits are sticky and are W1C. The bits in the `TIMER_STAT_ILAT` register provide information regarding each timer interrupt request source.

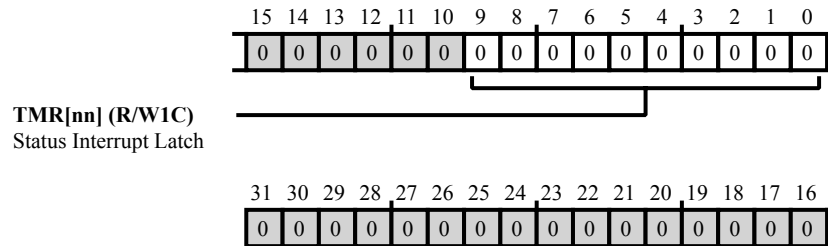


Figure 18-20: `TIMER_STAT_ILAT` Register Diagram

Table 18-32: `TIMER_STAT_ILAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W1C)	<code>TMR[nn]</code>	Status Interrupt Latch. For all <code>TIMER_STAT_ILAT.TMR[nn]</code> bits, status of 0 indicates no error interrupt request is latched, and status of 1 indicates a timer counter overflow or programming error interrupt request is latched.

Status Interrupt Mask Register

While each timer may generate a status interrupt request, these requests are OR'ed to generate a single status interrupt signal to the system event controller. The `TIMER_STAT_IMSK` register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the `TIMER_STAT_IMSK` register is `0xFFFF`, masking these interrupts after reset.

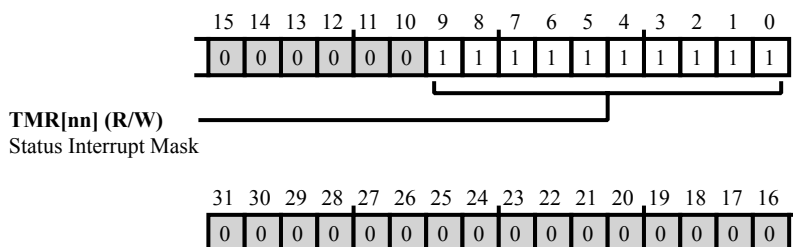


Figure 18-21: `TIMER_STAT_IMSK` Register Diagram

Table 18-33: `TIMER_STAT_IMSK` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	<code>TMR[nn]</code>	Status Interrupt Mask. For all <code>TIMER_STAT_IMSK.TMR[nn]</code> bits, write =0 unmask (enables) the corresponding status interrupt request, and write =1 masks (disables) the corresponding status interrupt request.

Stop Configuration Register

The `TIMER_STOP_CFG` register selects the stop mode for each timer. Timers may be stopped abruptly (immediate halt - all modes) or gracefully in PWMOUT modes (single pulse and continuous). The halt is achieved through either a write =0 to the corresponding bit in `TIMER_RUN` or a write =1 to the corresponding bit in `TIMER_RUN_CLR`. A read of `TIMER_STOP_CFG` returns the last value written.

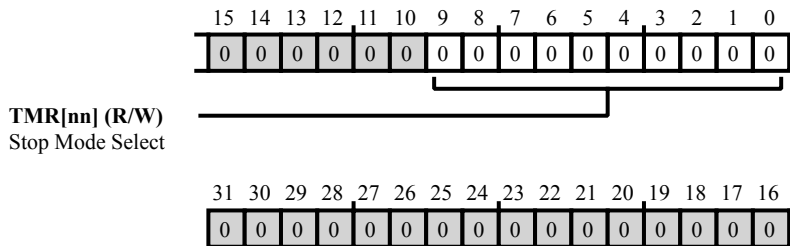


Figure 18-22: `TIMER_STOP_CFG` Register Diagram

Table 18-34: `TIMER_STOP_CFG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	<code>TMR[nn]</code>	Stop Mode Select. For all <code>TIMER_STOP_CFG.TMR[nn]</code> bits, write =0 for graceful termination (PWMOUT modes only), and write =1 for abrupt (immediate halt) on stop.

Stop Configuration Clear Register

This is an alias register, providing a mechanism to clear a specific bit in the `TIMER_STOP_CFG` register without affecting other bits in `TIMER_STOP_CFG`. To clear a bit in `TIMER_STOP_CFG`, software must write a 1 to the corresponding bit of `TIMER_STOP_CFG_CLR` register. Writing a zero has no effect. Because the `TIMER_STOP_CFG_CLR` register is a write-only register, the result of any write to this register must be checked by reading the `TIMER_STOP_CFG` register. A read of the `TIMER_STOP_CFG_CLR` register returns 0x0000.

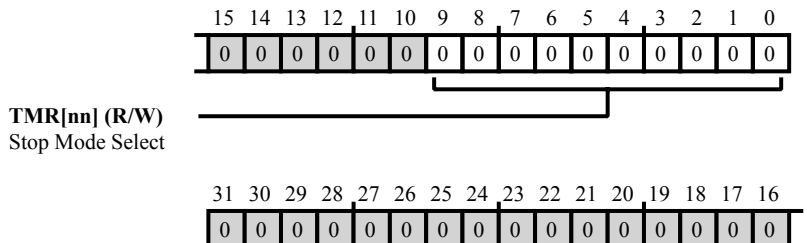


Figure 18-23: TIMER_STOP_CFG_CLR Register Diagram

Table 18-35: TIMER_STOP_CFG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R0/W1C)	TMR[nn]	STOP_CFG Clear Alias. For all <code>TIMER_STOP_CFG_CLR.TMR[nn]</code> bits, write =0 has no effect, and write =1 for graceful stop in PWMOUT modes (clearing the corresponding stop mode select bit in the <code>TIMER_STOP_CFG</code> register). Using <code>TIMER_STOP_CFG_CLR</code> to clear stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Stop Configuration Set Register

This is an alias register, providing a mechanism to set a specific bit in the `TIMER_STOP_CFG` register without affecting other bits in `TIMER_STOP_CFG`. To set a bit in the `TIMER_STOP_CFG` register, software must write a 1 to the corresponding bit of the `TIMER_STOP_CFG_SET` register. Writing a zero has no effect. Because the `TIMER_STOP_CFG_SET` register is a write-only register, the result of any write to this register must be checked by reading the `TIMER_STOP_CFG` register. A read of the `TIMER_STOP_CFG_SET` register returns 0x0000.

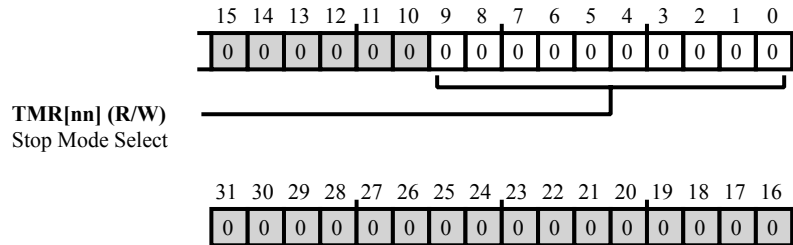


Figure 18-24: `TIMER_STOP_CFG_SET` Register Diagram

Table 18-36: `TIMER_STOP_CFG_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R0/W1S)	TMR[nn]	<p>STOP_CFG Set Alias.</p> <p>For all <code>TIMER_STOP_CFG_SET.TMR[nn]</code> bits, write =0 has no effect, and write =1 for abrupt stop (setting the corresponding stop mode select bit in the <code>TIMER_STOP_CFG</code> register). Using <code>TIMER_STOP_CFG_SET</code> to set stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.</p>

Timer n Configuration Register

Each timer has a `TIMER_TMR[n]_CFG` register that specifies its operating mode. Only write to a `TIMER_TMR[n]_CFG` register when the corresponding timer is not running.

After disabling a timer operating in PWMOUT mode, verify that the timer has stopped running by checking the start/stop status of the timer in the `TIMER_RUN` register before writing to the timer's `TIMER_TMR[n]_CFG` register.

Note that a timer's `TIMER_TMR[n]_CFG` register may be read at any time.

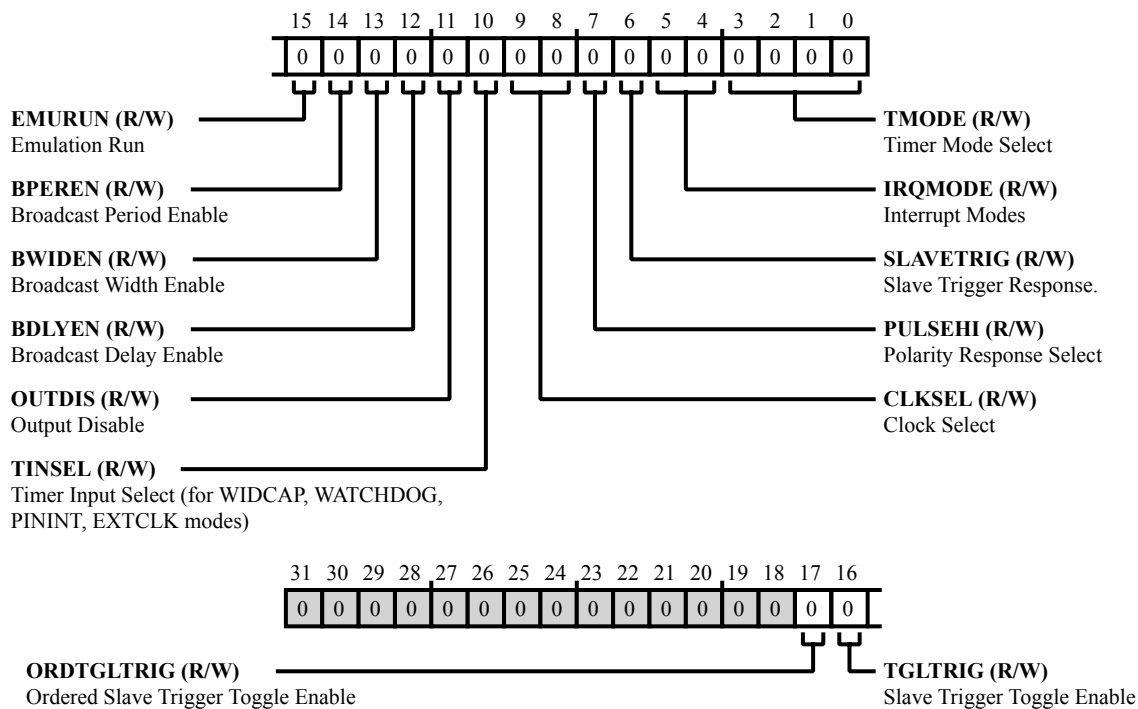


Figure 18-25: `TIMER_TMR[n]_CFG` Register Diagram

Table 18-37: TIMER_TMR[n]_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	ORDTGLTRIG	Ordered Slave Trigger Toggle Enable. The <code>TIMER_TMR[n]_CFG.ORDTGLTRIG</code> bit controls ordered slave trigger toggle mode. The timer(slave) state is not toggled by writes to the <code>TIMER_RUN</code> , <code>TIMER_RUN_SET</code> and <code>TIMER_RUN_CLR</code> registers when this bit is enabled.
		0 No effect
		1 Trigger pulse on Trigger Slave 0 starts the timer if it in halt state and trigger pulse in Trigger Slave 1 stops the timer if it is running
16 (R/W)	TGLTRIG	Slave Trigger Toggle Enable. The <code>TIMER_TMR[n]_CFG.TGLTRIG</code> bit stops the timer if it is running and starts the timer if it is halted (in the stop state). If the <code>TIMER_TMR[n]_CFG.TGLTRIG</code> bit is set, then the setting of the <code>TIMER_TMR[n]_CFG.SLAVETRIG</code> bit is ignored.
		0 Slave Trigger Response Depends on SLAVETRIG Bit Setting
		1 Slave Trigger Toggles Timer State
15 (R/W)	EMURUN	Emulation Run. The <code>TIMER_TMR[n]_CFG.EMURUN</code> bit causes the timer to run (count) during emulation.
		0 Stop Timer During Emulation
		1 Run Timer During Emulation
14 (R/W)	BPEREN	Broadcast Period Enable. The <code>TIMER_TMR[n]_CFG.BPEREN</code> bit enables updates to the <code>TIMER_TMR[n]_PER</code> register simultaneously across more than one timer.
		0 Disable Broadcast to PER Register
		1 Enable Broadcast to PER Register
13 (R/W)	BWIDEN	Broadcast Width Enable. The <code>TIMER_TMR[n]_CFG.BWIDEN</code> bit enables updates to the <code>TIMER_TMR[n]_WID</code> register simultaneously across more than one timer.
		0 Disable Broadcast to WID Register
		1 Enable Broadcast to WID Register

Table 18-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	BDLYEN	Broadcast Delay Enable. The <code>TIMER_TMR[n]_CFG.BDLYEN</code> bit enables updates to the <code>TIMER_TMR[n]_DLY</code> register simultaneously across more than one timer.
		0 Disable Broadcast to DLY Register
		1 Enable Broadcast to DLY Register
11 (R/W)	OUTDIS	Output Disable. The <code>TIMER_TMR[n]_CFG.OUTDIS</code> bit enables or disables the timer pin output buffer.
		0 Enable TMR Pin Output Buffer
		1 Disable TMR Pin Output Buffer
10 (R/W)	TINSEL	Timer Input Select (for WIDCAP, WATCHDOG, PININT, EXTCLK modes).
		0 Use TMR Pin Input
		1 Use TMR Alternate Capture Input
9:8 (R/W)	CLKSEL	Clock Select. The <code>TIMER_TMR[n]_CFG.CLKSEL</code> bit field selects the TIMER clock to use.
		0 Use SCLK
		1 Use TMR_ALT_CLK0 as TMR Clock
		3 Use TMR_ALT_CLK1 as TMR Clock
7 (R/W)	PULSEHI	Polarity Response Select. The <code>TIMER_TMR[n]_CFG.PULSEHI</code> bit defines specific behaviors of the timer based on the operating mode. For more information, see the specific operating mode in the Programming Concepts section.
		0 Negative Response or Pulse. A Negative Edge Response or Negative Action Pulse on the TMR pin.
		1 Positive Response or Pulse. A Positive Edge Response or Positive Action Pulse on the TMR pin.
6 (R/W)	SLAVETRIG	Slave Trigger Response.. The <code>TIMER_TMR[n]_CFG.SLAVETRIG</code> bit controls the trigger response. The trigger pulse has no effect (to stop or start the timer) if the timer is already in the requested state.
		0 Pulse Stops Timer if it is Running
		1 Pulse Starts Timer if it is Stopped

Table 18-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	IRQMODE	<p>Interrupt Modes.</p> <p>The <code>TIMER_TMR[n]_CFG</code>. <code>IRQMODE</code> bit field selects the interrupt request mode. Note that any mismatched combination of the <code>TIMER_TMR[n]_CFG</code>. <code>IRQMODE</code> and the <code>TIMER_TMR[n]_CFG</code>. <code>TMODE</code> bits results in no interrupt being generated. In WIDCAP modes, the position of the interrupt is controlled with the <code>TIMER_TMR[n]_CFG</code>. <code>TMODE</code> bit, and the <code>TIMER_TMR[n]_CFG</code>. <code>IRQMODE</code> bit is ignored.</p> <p>Whenever an interrupt is generated, a trigger master pulse is also generated, if enabled in the <code>TIMER_TRG_MSK</code> register.</p>	
		0	Active Edge Mode. The timer generates an interrupt at every active edge. The active edge polarity depends on the state of the <code>TIMER_TMR[n]_CFG</code> . <code>PULSEHI</code> bit. Valid for PININT mode only.
		1	Delay Expired Mode. The timer generates an interrupt when the <code>TIMER_TMR[n]_CNT</code> value reaches the value in the <code>TIMER_TMR[n]_DLY</code> register. This mode is valid for all PWMOUT modes.
		2	Width Plus Delay Expired Mode. The timer generates an interrupt when the <code>TIMER_TMR[n]_CNT</code> value reaches the value in the <code>TIMER_TMR[n]_WID</code> register plus the value in the <code>TIMER_TMR[n]_DLY</code> register. (PWMOUT modes only)
		3	Period Expired Mode. The timer generates an interrupt when the <code>TIMER_TMR[n]_CNT</code> value reaches the value in the <code>TIMER_TMR[n]_PER</code> register. (Continuous PWMOUT and EXTCLK modes only)
3:0 (R/W)	TMODE	<p>Timer Mode Select.</p> <p>The <code>TIMER_TMR[n]_CFG</code>. <code>TMODE</code> bit field selects the operating mode of each timer.</p>	
		0-7	Idle Mode
		8	Period Watchdog Mode
		9	Width Watchdog Mode

Table 18-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Measurement Report at Asserting Edge of Waveform
		11	Measurement Report at Deasserting Edge of Waveform
		12	Continuous PWMOUT Mode
		13	Single Pulse PWMOUT Mode
		14	EXTCLK Mode
		15	PININT (pin interrupt) Mode

Timer n Counter Register

The `TIMER_TMR[n]_CNT` register holds the current timer count. After enabling, the count is re-initialized to either 0x0 or 0x1, depending on the configuration and mode. The `TIMER_TMR[n]_CNT` register is read-only and may be read at any time (whether the timer is running or stopped). Reading the `TIMER_TMR[n]_CNT` register returns an atomic 32-bit value.

Depending on the timer operation mode, the counter increment can be clocked by a number of sources, including SCLK, the TMR or alternate capture input pins, `ACLK[n]`. The counter retains its value after the timer is disabled.

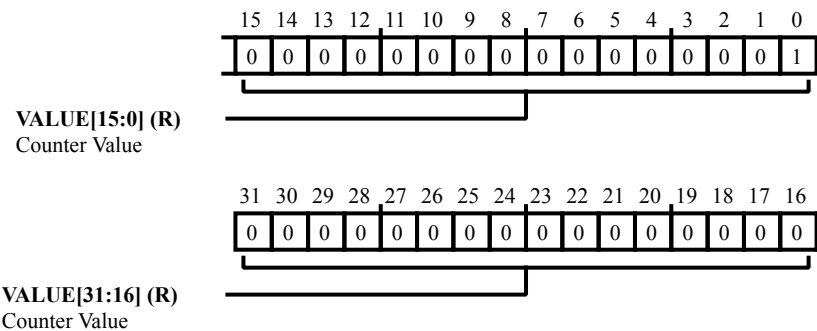


Figure 18-26: `TIMER_TMR[n]_CNT` Register Diagram

Table 18-38: `TIMER_TMR[n]_CNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value. The <code>TIMER_TMR[n]_CNT.VALUE</code> bit field holds the current timer count.

Timer n Delay Register

The `TIMER_TMR[n]_DLY` register holds the delay value for the corresponding timer. This register's use is based on the selected timer mode.

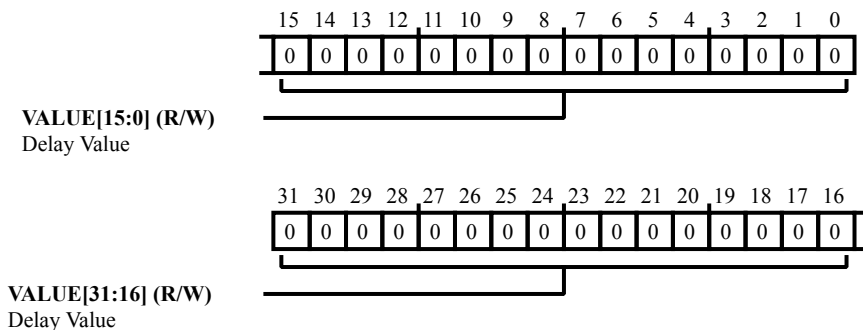


Figure 18-27: `TIMER_TMR[n]_DLY` Register Diagram

Table 18-39: `TIMER_TMR[n]_DLY` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Value. The <code>TIMER_TMR[n]_DLY.VALUE</code> bit field holds the delay value for the corresponding timer.

Timer n Period Register

The `TIMER_TMR[n]_PER` register holds the period value for the corresponding timer. This register's use is based on the selected timer mode.

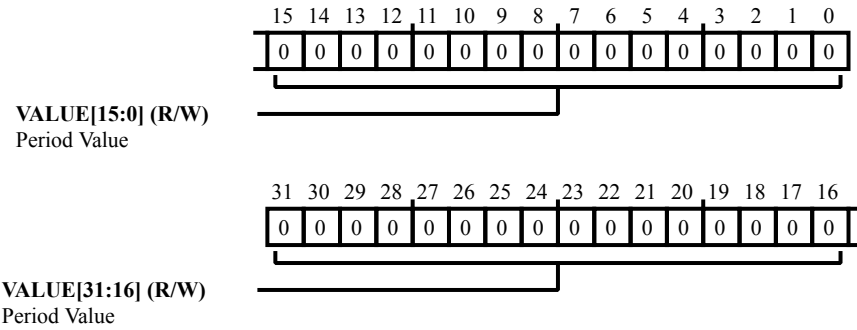


Figure 18-28: `TIMER_TMR[n]_PER` Register Diagram

Table 18-40: `TIMER_TMR[n]_PER` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Period Value. The <code>TIMER_TMR[n]_PER.VALUE</code> bit field holds the period value for the corresponding timer.

Timer n Width Register

The `TIMER_TMR[n]_WID` register holds the width value for the corresponding timer. This register's use is based on the selected timer mode.

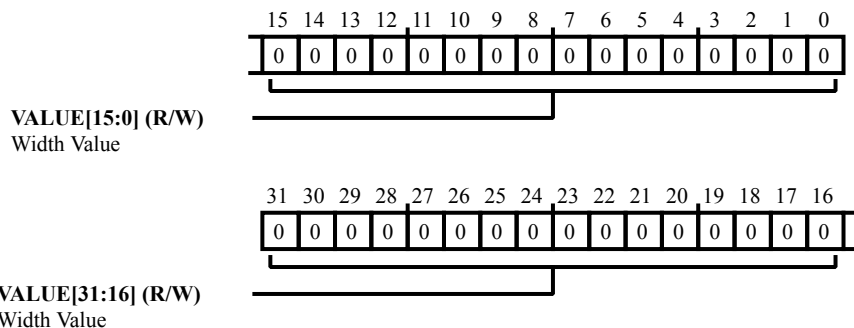


Figure 18-29: `TIMER_TMR[n]_WID` Register Diagram

Table 18-41: `TIMER_TMR[n]_WID` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Width Value. The <code>TIMER_TMR[n]_WID.VALUE</code> bit field holds the width value for the corresponding timer.

Trigger Slave Enable Register

As a trigger slave, each timer can generate a unique data trigger pulse signal. The `TIMER_TRG_IE` contains trigger input enable bits for these signals, disabling or enabling the triggers as programmed. The reset value of the `TIMER_TRG_IE` register is `0xFFFF`, masking these triggers after reset.

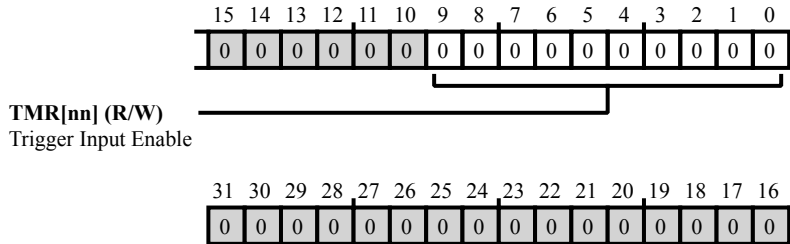


Figure 18-30: `TIMER_TRG_IE` Register Diagram

Table 18-42: `TIMER_TRG_IE` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	<code>TMR[nn]</code>	Trigger Input Enable. For all <code>TIMER_TRG_IE.TMR[nn]</code> bits, write =0 disables the corresponding trigger input, and write =1 enables the corresponding trigger input.

Trigger Master Mask Register

As a trigger master, each timer can generate a unique data trigger pulse signal. The `TIMER_TRG_MSK` register contains a trigger mask for these outputs, masking (disabling) or unmasking (enabling) the triggers as programmed. The reset value of the `TIMER_TRG_MSK` register is `0xFFFF`, masking these triggers after reset.

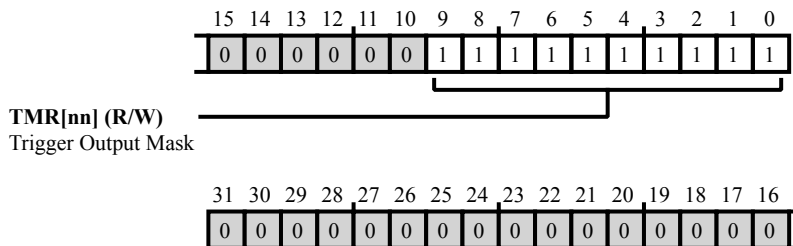


Figure 18-31: `TIMER_TRG_MSK` Register Diagram

Table 18-43: `TIMER_TRG_MSK` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	TMR[nn]	Trigger Output Mask. For all <code>TIMER_TRG_MSK.TMR[nn]</code> bits, write =0 unmask (enables) the corresponding data trigger output, and write =1 masks (disables) the corresponding data trigger output.

19 General-Purpose Counter (CNT)

The GP counter converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

The GP counter provides flexible ways to establish position information. When used with the GP timer block, the GP counter can allow for the acquisition of coherent position or time stamp information that enables speed calculation.

GP Counter Features

The GP counter includes the following features:

- 32-bit up or down counter
- Quadrature encode mode (Gray code)
- Binary encoder mode
- Alternative frequency-direction mode
- Timed direction and up or down counting modes
- Zero marker or push-button support
- Capture event timing in association with GP Timer
- Boundary comparison and boundary setting features

GP Counter Functional Description

The *GP Counter Block Diagram* shows a block diagram of the GP counter. The CNT_UD and CNT_DG pins accept various forms of incremental inputs. The 32-bit counter processes the inputs. The GP counter uses the CNT_ZM pin to sense the pressing of a push button.

NOTE: When enabled, the GP counter requires 3 SCLK cycles of initialization before recognizing valid toggles on its input pins.

The three input pins can be filtered (debounced) before the GP counter evaluates them.

The GP counter features a flexible boundary comparison. In all of the operating modes, the counter can be compared to an upper and lower limit. It takes various actions when reaching these limits.

The GP counter has a flexible input selection. Apart from accepting inputs from the CNT0_UD and CNT0_DG and PORT pins, the counter can be configured to accept trigger inputs by setting PADS_PCFG0.CNT0UDSEL and PADS_PCFG0.CNT0DGSEL bits. Refer to the PADS_PCFG0 register description in General-Purpose Ports (PORT) chapter for details.

The module can optionally generate an interrupt request to the system through its IRQ line. On many processors, a GP timer module can use an output to generate time stamps on certain events.

ADSP-2156x CNT Register List

The GP Counter (CNT) provides support for manually controlled rotary controllers, such as the volume wheel on a radio device. This unit also supports industrial encoders.

The CNT converts pulses from incremental position encoders into data that is representative of the actual position. To complete this task, the CNT integrates (counts) pulses on one or two inputs. Because integration provides relative position, a zero position input (zero marker) is usually provided that establishes a reference point, verifying that the acquired position does not drift over time. The incremental position information may also be used to determine speed, if the relevant time intervals are measured. The CNT provides flexible ways to establish position information. When used in conjunction with the General-purpose Timer (TIMER), the CNT allows acquisition of coherent position/time stamp information, enabling speed calculation.

A set of registers govern CNT operations. For more information on CNT functionality, see the CNT register descriptions.

Table 19-1: ADSP-2156x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MIN	Minimum Count Register
CNT_STAT	Status Register

ADSP-2156x CNT Interrupt List

Table 19-2: ADSP-2156x CNT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
108	CNT0_STAT	CNT0 Status	Level	

ADSP-2156x CNT Trigger List

Table 19-3: ADSP-2156x CNT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
16	CNT0_STAT	CNT0 Status	Level
98	CNT0_UD	CNT0 CNT0 Count Up and Direction	Level
99	CNT0_DG	CNT0 CNT0 Count Down and Gate	Level
100	CNT0_TO	CNT0 CNT0 Output to Timer Block	Level

Table 19-4: ADSP-2156x CNT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
114	CNT0_UD	CNT0 CNT0 Count Up and Direction Input	Pulse
115	CNT0_DG	CNT0 CNT0 Count Down and Gate Input	Pulse

GP Counter Operating Modes

The GP counter has the following five modes of operation.

1. Quadrature Encoder
2. Binary Encoder
3. Up/Down Counter
4. Direction Counter
5. Timed Direction

With the exception of the timed direction mode, the GP counter can operate with the GP timer block to capture additional timing information (time stamps) associated with events detected by this block.

Quadrature Encoder Mode

In this mode, the CNT_UD and CNT_DG inputs expect a quadrature-encoded signal that is interpreted as a two-bit Gray code. The order of transitions of the CNT_UD and CNT_DG inputs determines whether the counter increments or decrements. The CNT_CNTR register contains the number of transitions that have occurred as shown in the *Quadrature Events and Counting Mechanism* table. Optionally, an interrupt is generated when both inputs

change within one SCLK cycle. Gray coding prohibits such transitions. Therefore, the `CNT_CNTR` register remains unchanged, and an error condition is signaled.

Table 19-5: Quadrature Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG, CUD Inputs	00	01	11	10	00	01	11	10	00

It is possible to reverse the count direction of the Gray-coded signal by enabling the polarity inverter of either the `CNT_UD` pin or the `CNT_DG` pin. Inverting both pins does not alter the behavior. The GP counter can enable this feature with the `CNT_CFG.CDGINV` and `CNT_CFG.CUDINV` bits.

As an example, the `CNT_DG` and `CNT_UD` inputs are 00 and the next transition is to 01. These inputs normally change the counter in increments as shown in the table. If the `CNT_UD` polarity is inverted, this condition generates a received input of 01 followed by 00. The results is a decrement of the counter, altering the behavior of the connected hardware.

Binary Encoder Mode

This mode is almost identical to quadrature encoder mode, with the exception that the `CNT_UD`: `CNT_DG` inputs expect a binary-encoded signal. The order of transitions of the `CNT_UD` and `CNT_DG` inputs determines whether the counter increments or decrements. The `CNT_CNTR` register contains the number of transitions that have occurred as shown in the *Binary Events and Counting Mechanism* table. Optionally, an interrupt is generated when the detected code steps by more than 1 (in binary arithmetic) within one SCLK cycle. Such transitions are erroneous. Therefore, the `CNT_CNTR` register remains unchanged, and an error condition is signaled.

Table 19-6: Binary Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG:CUD Inputs	00	01	10	11	00	01	10	11	00

Reversing the `CNT_UD` and `CNT_DG` pin polarity has a different effect in binary encoder mode than for the quadrature encoder mode. Inverting the polarity of the `CNT_UD` pin only, or inverting both the `CNT_UD` and `CNT_DG` pins, results in reversing the count direction.

Up/Down Counter Mode

In this mode, the counter increments or decrements at every active edge of the input pins. The GP counter uses the `CNT_CFG.CUDINV` bit to select an active edge and has the following results.

- If the GP counter module detects an active edge at the `CNT_UD` input, the counter increments.
- If the GP counter module detects an active edge at the `CNT_DG` input, the counter decrements.
- If simultaneous edges occur on the `CNT_DG` and `CNT_UD` pins, the counter remains unchanged, and both up-count and down-count events are signaled in the `CNT_STAT` register.

Direction Counter Mode

In this mode, the counter is incremented or decremented at every active edge of the `CNT_DG` input pin. The state of the `CNT_UD` input determines whether the counter increments or decrements. The GP counter uses the `CNT_CFG.CUDINV` bit to select the polarity.

If the GP counter detects an active edge at the `CNT_DG` input, the counter value changes by one in the selected direction.

Timed Direction Mode

In this mode, the counter is incremented or decremented at each `SCLK` cycle. The state of the `CNT_UD` input determines whether the counter increments or decrements. The GP counter uses the `CNT_CFG.CUDINV` bit to select the polarity. The `CNT_DG` pin can be used to gate the clock. The GP counter uses the `CNT_CFG.CDGINV` bit to select the polarity.

GP Counter Programming Model

The following sections provide information for programming the interface.

GP Counter General Programming Flow

The following are general guidelines for configuring and enabling the GP counter.

1. Initialize (but do not enable) the GP counter for the desired mode and settings through the `CNT_CFG` register.
2. Usually, events of interest are processed using interrupts rather than by polling status bits. In this case, clear all status bits and activate the interrupt generation requests with the `CNT_IMSK` register.
3. Configure interrupts at the system level to insure desired interrupt signaling to the system.
4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupt requests and the GP Counter itself using the `CNT_IMSK` and `CNT_CFG` registers, respectively.

GP Counter Mode Configuration

The GP counter can use the `CNT_ZM` input pin to sense the zero marker output of a rotary device or to detect the pressing of a push button. There are four programming schemes, which are functional in all counter modes:

- Push-button mode
- Zero-marker-zeros-counter mode
- Zero-marker-error mode
- Zero-once mode

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set `CNT_IMSK.CZM` to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the `CNT_ZM` input sets the `CNT_IMSK.CZM` bit.

Configuring Zero-Marker-Zeros-Counter Mode

The following provides information on configuring zero-marker-zeros-counter mode for the GP counter.

1. Set `CNT_IMSK.CZMZ` to enable `CNT_CNTR`. The zero marker interrupt zeroes the counter.
2. Set `CNT_CFG.ZMZC` to enable ZMZC mode.
3. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
4. Proceed with any other desired configuration steps and enable the peripheral.

This configuration causes an active level at the `CNT_ZM` pin to clear the `CNT_CNTR` register and keep it cleared until the `CNT_ZM` pin is deactivated. In addition, the `CNT_STAT.CZMZ` bit is set.

Configuring Zero-Marker-Error Mode

The GP counter uses this mode to detect discrepancies between counter-value and the zero marker output of certain rotary encoder devices.

1. Set the `CNT_STAT.CZME` bit to enable this mode.
2. Select the active edge of the `CNT_ZM` pin through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

When the GP counter detects an active edge at the `CNT_ZM` input pin, it compares the four LSBs of the `CNT_CNTR` register to zero. If they are not zero, the GP counter uses `CNT_STAT.CZME` bit to signal a mismatch.

Configuring Zero-Once Mode

The GP counter uses this mode to perform an initial reset of the counter-value when it detects an active zero marker. After that, the zero marker is ignored (the counter is no longer reset).

1. Set the `CNT_CMD.W1ZMONCE` bit to enable this mode.
2. Select the active edge of the `CNT_ZM` pin through the `CNT_CFG.CZMINV` bit.
3. Ensure that at least one of the following bits is enabled: `CNT_IMSK.CZM`, `CNT_IMSK.CZME`, `CNT_IMSK.CZMZ`.

4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_CNTR` register and the `CNT_CMD.W1ZMONCE` bit are cleared on the next active edge of the `CNT_ZM` pin. Now the `CNT_CMD.W1ZMONCE` bit can be read to check whether the event has already occurred.

Configuring Boundary Auto-Extend Mode

In this mode, hardware modifies the boundary registers (`CNT_MIN` and `CNT_MAX`) whenever the `CNT_CNTR` value reaches either of them. The GP counter uses this mode to monitor the widest angle a thumb wheel even if the software did not generate interrupts.

1. Initialize `CNT_CNTR` with the desired value.
2. Set both `CNT_MIN` and `CNT_MAX` to this same value.
3. Configure the `CNT_CFG.BNDMODE` field for auto extend mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_MAX` register is loaded with the current `CNT_CNTR` value when the latter increments beyond the `CNT_MAX` value. Similarly, the `CNT_MIN` register is loaded with the `CNT_CNTR` value when the latter decrements below the `CNT_MIN` value. The `CNT_STAT.MAXC` and `CNT_STAT.MINC` status bits are set when the `CNT_CNTR` value matches the respective boundary register value.

Configuring Boundary Capture Mode

In this mode, the `CNT_CNTR` value is latched into the `CNT_MIN` register at one detected edge of the `CNT_ZM` input pin, and latched into the `CNT_MAX` boundary register at the opposite edge.

1. To capture the `CNT_ZM` pin rising edge into `CNT_MIN` and the falling edge into `CNT_MAX`, program `CNT_CFG.CZMINV` for active high polarity. Conversely, to capture the `CNT_ZM` pin falling edge into `CNT_MIN` and the rising edge into `CNT_MAX`, program `CNT_CFG.CZMINV` for active low polarity.
2. Program the `CNT_IMSK.MAXC` and `CNT_IMSK.MINC` interrupt mask bits according to interrupt generation requirements.
3. Configure the `CNT_CFG.BNDMODE` field for boundary capture mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_STAT.MAXC` and `CNT_STAT.MINC` status bits report the capture event, depending on how interrupt masks are configured.

Configuring Boundary Compare and Boundary Zero Modes

In these modes, the two boundary registers (`CNT_MAX` and `CNT_MIN`) are compared to the value in the `CNT_CNTR` register.

1. Program `CNT_MAX` and `CNT_MIN` registers with appropriate upper and lower range values.

2. Program the `CNT_IMSK.MAXC` and `CNT_IMSK.MINC` interrupt mask bits according to interrupt generation requirements.
3. Configure the `CNT_CFG.BNDMODE` field for boundary compare mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

If after incrementing, `CNT_CNTR = CNT_MAX`, then the `CNT_STAT.MAXC` bit is set. Similarly if after decrementing, `CNT_CNTR = CNT_MIN`, then the `CNT_STAT.MINC` bit is set.

Additionally, for boundary zero mode, the counter-value in `CNT_CNTR` is set to zero. The `CNT_STAT.MAXC` and `CNT_STAT.MINC` bits are not set when software updates the `CNT_MAX` or `CNT_MIN` registers.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set `CNT_IMSK.CZM` to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the `CNT_ZM` input sets the `CNT_IMSK.CZM` bit.

GP Counter Programming Concepts

Using the features, operating modes, and event control for the GP counter to their greatest potential requires an understanding of some GP counter-related concepts. Some key aspects to consider are input noise filtering and capturing timing information.

CNT Input Noise Filtering

In all modes, the three input pins can be filtered to present clean signals to the GP counter logic. The GP counter uses the `CNT_CFG.DEBEN` bit to enable or disable this filtering. The *Programmable Noise Filtering* figure shows the filtering operation for the `CNT_UD` pin.

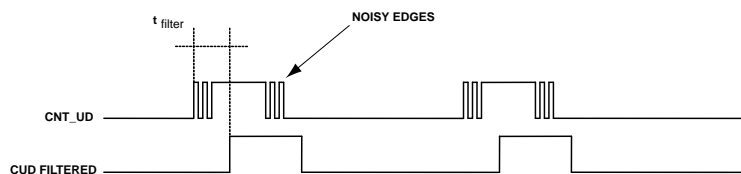


Figure 19-1: Programmable Noise Filtering

The CNT module implements the filtering mechanism using counters for each GP counter pin, where each counter is initialized from the `CNT_DEBNCE.DPRESCALE` field. When a transition is detected on a pin, the corresponding counter starts counting up to the programmed number of SCLK cycles. The state of the pin is latched after time t_{filter} and passed on to the GP counter logic.

The following formula determines the time t_{filter} , given SCLK and the CNT_DEBNCE.DPRESCALE value, where lower values of CNT_DEBNCE.DPRESCALE result in shorter debounce delays:

$$t_{\text{filter}} = 128 \times (2^{\text{DPRESCALE}} \times \text{SCLK})$$

Capturing Counter Interval and CNT_CNTR Read Timing

When the count speed is low, it is often useful to capture the time elapsed since the last count event. Program the `TIMER_TMR[n]_CFG` register of the associated GP timer in a width capture mode with the following bit settings.

- `TIMER_TMR[n]_CFG.PULSEHI = 0`
- `TIMER_TMR[n]_CFG.TMODE = b#1011`
- `TIMER_TMR[n]_CFG.TINSEL = 1`

The *Capture Intervals* figure shows and the following list describe the operation of the GP counter and the GP timer in this mode.

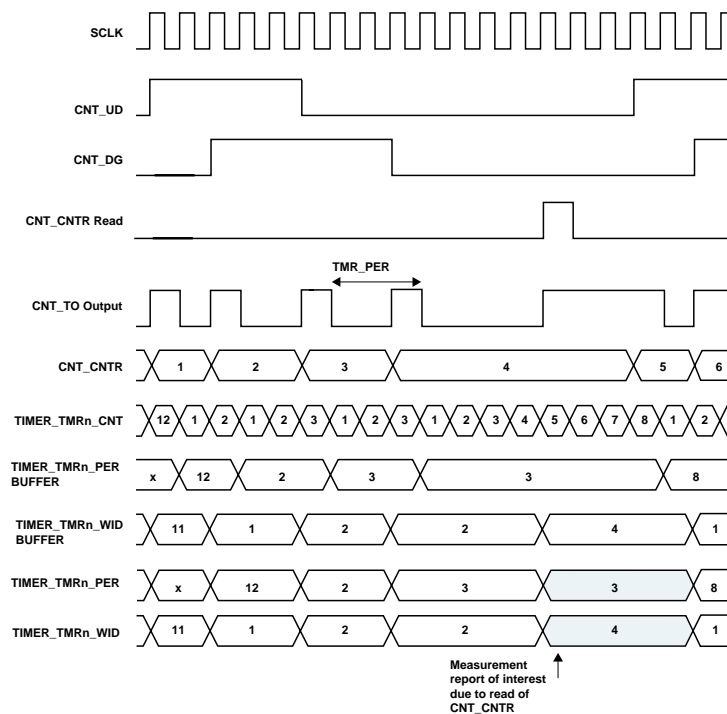


Figure 19-2: Capture Intervals

NOTE: SCLK in the *Capture Intervals* figure is SCLK.

1. The `CNT_TO` signal generates a pulse every time a count event occurs. In addition, when the processor reads the `CNT_CNTR` register, the `CNT_TO` signal presents a pulse which is extended (high) until the next count event.
2. The GP timer updates its `TIMER_TMR[n]_PER` register with the period (measured from falling edge to falling edge, because `TIMER_TMR[n]_CFG.PULSEHI = 0`) of the `CNT_TO` signal.

3. The `TIMER_TMR[n]_WID` register is updated with the pulse width (the portion where `CNT_TO` is low, again because `TIMER_TMR[n]_CFG.PULSEHI = 0`).
4. Both registers are updated at every rising edge of the `CNT_TO` signal (because `TIMER_TMR[n]_CFG.TMODE = b#011`).

The `TIMER_TMR[n]_PER` register contains the period between the last two count events. The `TIMER_TMR[n]_WID` register contains the time since the last count event and the read of the `CNT_CNTR` register, both measured in SCLK cycles.

Read the `CNT_CNTR` register to latch the two time measurements, providing a coherent triplet of information to calculate speed and position.

NOTE: Speed restrictions apply to the use of the `CNT_TO` signal. Therefore, programs must not operate at count event rates that are high. For instance, if `CNT_CNTR` is incremented or decremented every SCLK cycle (timed direction mode), the `CNT_TO` signal is not valid.

Capturing Time Interval Between Successive Counter Events

When the required timing information is the interval between successive count events, program the associated timer in a width capture mode. Set the `TIMER_TMR[n]_CFG` bit of `TIMER_TMR[n]_CFG.PULSEHI = 1`, `TIMER_TMR[n]_CFG.TMODE = b#1010` and `TIMER_TMR[n]_CFG.TINSEL = 1`. Typically, this information is sufficient if the speed of GP counter events does not to reach low values.

The *Period Register Timing* figure shows the operation of the GP counter and the GP timer in this mode.

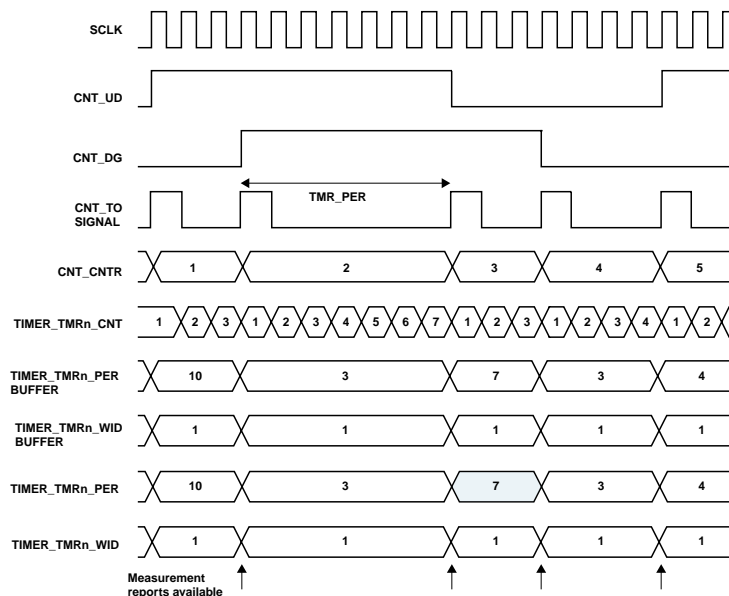


Figure 19-3: Period Register Timing

The `CNT_TO` signal generates a pulse every time a count event occurs. The GP timer updates its `TIMER_TMR[n]_PER` register with the period (measured from rising edge to rising edge) of the `CNT_TO` signal.

The `TIMER_TMR[n]_PER` register is updated at every rising edge of the `CNT_TO` signal and contains the number of `SCLK` cycles that have elapsed since the previous rising edge.

Incidentally, the `TIMER_TMR[n]_WID` register is also updated at the same time, but is generally of no interest in this mode of operation. If no reads of the `CNT_CNTR` register occur between counter events, the `TIMER_TMR[n]_WID` register only contains the width of the `CNT_TO` pulse. If a read of `CNT_CNTR` has occurred between events, the `TIMER_TMR[n]_WID` register contains the time between the read of `CNT_CNTR` and the next event.

This mode can also be used with `TIMER_TMR[n]_CFG.PULSEHI = 0`. In this case, the period of `CNT_TO` is measured between falling edges. It results in the same values as in the previous case, only the latching occurs one `SCLK` cycle later.

GP Counter Event Control

Eleven events can be signaled to the processor using status information and optional interrupt requests. The GP counter uses the respective bits in the `CNT_IMSK` register to enable the interrupt requests. It uses dedicated bits in the `CNT_STAT` register to report events. When an interrupt request from the GP counter is serviced, the application software is responsible for correct interpretation of the events. It is recommended to logically AND the content of the `CNT_IMSK` and `CNT_STAT` registers to identify pending interrupt requests.

Perform a write-one-to-clear (W1C) operation to the `CNT_STAT` register to clear the interrupt requests. Hardware does not clear the status bits automatically, unless the counter module is disabled.

The following sections describe the events associated with the GP counter.

Illegal Gray and Binary Code Events

When illegal transitions occur in quadrature encoder or binary encoder modes, the `CNT_STAT.IC` bit is set. If enabled by the `CNT_STAT.IC` bit, the counter module generates an interrupt request. Set the `CNT_STAT.IC` bit only in the quadrature encoder or binary encoder modes.

Up/Down Count Events

The GP counter uses the `CNT_STAT.UC` bit to indicate whether the counter has been incremented. Similarly, the `CNT_STAT.DC` bit reports decrements. The two events are independent. For instance, if the counter increments by one and then decrements by two, both bits remain set, even though the resulting counter-value shows a decrement by one.

In up/down counter mode, hardware can detect simultaneous active edges on the `CNT_UD` and `CNT_DG` inputs. In that case, the `CNT_CNTR` remains unchanged, but both the `CNT_STAT.UC` and `CNT_STAT.DC` bits are set. Interrupt requests for these events can be enabled through the `CNT_IMSK.UC` and `CNT_IMSK.DC` bits. Use this feature carefully when the counter is clocked at high rates. This suggestion is especially critical when the counter operates in `DIR_TMR` mode, as interrupts are generated every `SCLK` cycle.

These events can also be used for more push buttons, when GP counter features are unnecessary. When up/down counter mode is enabled, the GP counter can use these count events to report interrupts from push buttons that connect to the CNT_UD and CNT_DG inputs.

Zero-Count Events

The CNT_STAT.CZERO status bit indicates that the CNT_CNTR has reached a value equal to 0x0000 0000 after an increment or decrement. This bit is not set when the counter value is set to zero by a write to CNT_CNTR or by setting the CNT_CMD.W1LCNTZERO bit. If enabled by the CNT_IMSK.CZERO bit, the GP counter module generates an interrupt request.

Overflow Events

There are two status bits that indicate whether the signed counter-register has overflowed from a positive to a negative value or conversely. The CNT_STAT.COV31 bit reports that the 32-bit CNT_CNTR register has either incremented from 0x7FFF FFFF to 0x8000 0000, or decremented from 0x8000 0000 to 0x7FFF FFFF.

If enabled by the CNT_IMSK.COV31 bit, an interrupt request is generated. Similarly, in applications where only the lower 16 bits of the counter are of interest, the CNT_STAT.COV15 status bit reports counter transitions from 0xFFFF 7FFF to 0xFFFF 8000, or from 0xFFFF 8000 to 0xFFFF 7FFF. If enabled by the CNT_IMSK.COV15 bit, an interrupt request is generated.

Boundary Match Events

The CNT_STAT.MINC and CNT_STAT.MAXC status bits report boundary events as described in [Configuring Boundary Capture Mode](#). These bits are not set if the software updates the CNT_CNTR, CNT_MAX, or CNT_MIN registers or writes to the CNT_CMD register. The CNT_IMSK.MINC and CNT_IMSK.MAXC bits enable interrupt request generation on boundary events.

Zero Marker Events

The CNT_STAT.CZM, CNT_STAT.CZME, and CNT_STAT.CZMZ bits are associated with zero marker events, as described in [Configuring GP Counter Push-Button Operation](#). Each of these events can optionally generate an interrupt request, when enabled by the corresponding CNT_IMSK.CZM, CNT_IMSK.CZME and CNT_IMSK.CZMZ bits.

ADSP-2156x CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 19-7: ADSP-2156x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register

Table 19-7: ADSP-2156x CNT Register List (Continued)

Name	Description
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MIN	Minimum Count Register
CNT_STAT	Status Register

Configuration Register

The `CNT_CFG` register configures counter modes, configures input pins, and enables the CNT.

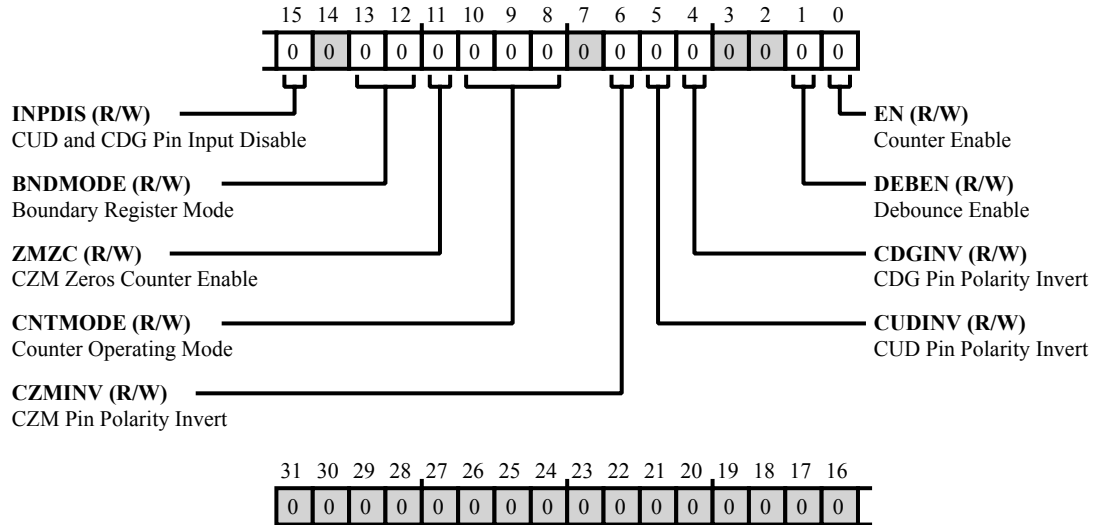


Figure 19-4: CNT_CFG Register Diagram

Table 19-8: CNT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	INPDIS	CUD and CDG Pin Input Disable. The <code>CNT_CFG</code> . <code>INPDIS</code> disables or enables the <code>CNT_UD</code> input pin and the <code>CNT_DG</code> pin.
		0 Enable
		1 Pin Input Disable
13:12 (R/W)	BNDMODE	Boundary Register Mode. The <code>CNT_CFG</code> . <code>BNDMODE</code> bit field selects the mode for the <code>CNT_MIN</code> and <code>CNT_MAX</code> boundary registers.
		0 BND_COMP. Boundary Compare Mode
		1 BND_ZERO. Boundary Zero Mode
		2 BND_CAPT. Boundary Capture Mode
		3 BND_AEXT. Boundary Auto-extend Mode

Table 19-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	ZMZC	CZM Zeros Counter Enable. The CNT_CFG.ZMZC bit enables or disables level sensitive - active CNT_ZM pin operation to zero the CNT_CNTR register.
		0 Disable
		1 Enable
10:8 (R/W)	CNTMODE	Counter Operating Mode. The CNT_CFG.CNTMODE bit field selects the operating mode for the CNT_UD input pin and the CNT_DG pin.
		0 QUAD_ENC. Quadrature Encoder Mode
		1 BIN_ENC. Binary Encoder Mode
		2 UD_CNT. Rotary Counter Mode
		4 DIR_CNT. Direction Counter Mode
5 DIR_TMR. Direction Timer Mode		
6 (R/W)	CZMINV	CZM Pin Polarity Invert. The CNT_CFG.CZMINV bit selects the polarity for the CNT_ZM pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.
		0 Active High, Rising Edge
		1 Active Low, Falling Edge
5 (R/W)	CUDINV	CUD Pin Polarity Invert. The CNT_CFG.CUDINV bit selects the polarity for the CNT_UD pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.
		0 Active High, Rising Edge
		1 Active Low, Falling Edge
4 (R/W)	CDGINV	CDG Pin Polarity Invert. The CNT_CFG.CDGINV bit selects the polarity for the CNT_DG pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.
		0 Active High, Rising Edge
		1 Active Low, Falling Edge

Table 19-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	DEBEN	Debounce Enable. The CNT_CFG.DEBEN bit enables or disables CNT input debounce filtering operation selected with the CNT_DEBNCE register.
		0 Disable
		1 Enable
0 (R/W)	EN	Counter Enable. The CNT_CFG.EN bit enables or disables CNT operation.
		0 Counter Disable
		1 Counter Enable

Command Register

The `CNT_CMD` register configures the CNT, enabling operations such as zeroing a counter register and copying or swapping boundary registers. These actions are taken by setting the appropriate bit.

Read operations from this register do not return meaningful values, with the exception of the `CNT_CMD.W1ZMONCE` bit, where a set bit indicates that the bit has been set by software before, but a zero marker event has not yet been detected on the `CNT_ZM` pin yet. For more information, see the CNT functional description.

The `CNT_CNTR`, `CNT_MIN`, and `CNT_MAX` registers can be initialized to zero by setting the `CNT_CMD.W1LCNTZERO`, `CNT_CMD.W1LMINZERO`, and `CNT_CMD.W1LMAXZERO` bits. In addition to clearing registers, the `CNT_CMD` register permits modifying the `CNT_MIN` and `CNT_MAX` boundary registers in a number of ways. The current counter value in the `CNT_CNTR` register can be captured and loaded into either of the two boundary registers to create new boundary limits. This operation is performed by setting the `CNT_CMD.W1LMAXCNT` and `CNT_CMD.W1LMINCNT` bits. Alternatively, the counter can be loaded from `CNT_MAX` or `CNT_MIN` using the `CNT_CMD.W1LCNTMAX` and `CNT_CMD.W1LCNTMIN` bits. It is also possible to transfer the current `CNT_MAX` value into `CNT_MIN` (or conversely) through the `CNT_CMD.W1LMINMAX` and `CNT_CMD.W1LMAXMIN` bits.

Another counter operation is the ability to only have the zero marker clear the `CNT_CNTR` register once. For more information, see the CNT functional description.

It is possible for multiple actions to be performed simultaneously by setting multiple bits in the `CNT_CMD` register. However, there are restrictions. The bits associated with each command have been grouped together such that all bits that involve a write to the `CNT_CNTR`, `CNT_MAX`, or `CNT_MIN` registers are located within bits 4-bit groups of the `CNT_CMD` register.

Note that a maximum of three commands can be issued at any one time, excluding the `CNT_CMD.W1ZMONCE` command. Also, note that `CNT_CMD.W1LCNTMIN`, `CNT_CMD.W1LCNTMAX`, and `CNT_CMD.W1LCNTZERO` bits have to be used exclusively. Never set more than one of them at the same time.

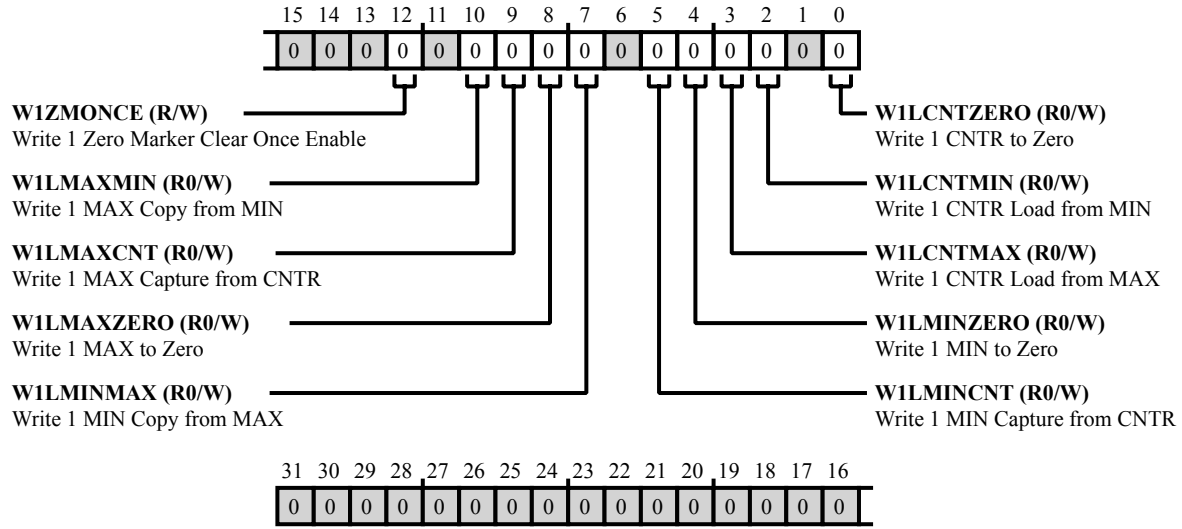


Figure 19-5: CNT_CMD Register Diagram

Table 19-9: CNT_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	W1ZMONCE	Write 1 Zero Marker Clear Once Enable. The CNT_CMD.W1ZMONCE enables a single zero marker clear of the CNT_CNTR register. Reading a 1 in this bit indicates that the bit has been set by software before, but no zero marker event has been detected on the CNT_ZM pin yet.
10 (R0/W)	W1LMAXMIN	Write 1 MAX Copy from MIN. The CNT_CMD.W1LMAXMIN bit transfers the current CNT_MIN register value into CNT_MAX register.
9 (R0/W)	W1LMAXCNT	Write 1 MAX Capture from CNTR. The CNT_CMD.W1LMAXCNT bit loads the current value in the CNT_CNTR register into the CNT_MAX register to create a new boundary limit.
8 (R0/W)	W1LMAXZERO	Write 1 MAX to Zero. Writing a 1 to the CNT_CMD.W1LMAXZERO bit clears the CNT_MAX register.
7 (R0/W)	W1LMINMAX	Write 1 MIN Copy from MAX. The CNT_CMD.W1LMINMAX bit transfers the current CNT_MAX register value into CNT_MIN register.
5 (R0/W)	W1LMINCNT	Write 1 MIN Capture from CNTR. The CNT_CMD.W1LMINCNT bit loads the current value in the CNT_CNTR register into the CNT_MIN register to create a new boundary limit.
4 (R0/W)	W1LMINZERO	Write 1 MIN to Zero. Writing a 1 to the CNT_CMD.W1LMINZERO bit clears the CNT_MIN register.

Table 19-9: CNT_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R0/W)	W1LCNTMAX	Write 1 CNTR Load from MAX. The CNT_CMD.W1LCNTMAX bit loads the current value in the CNT_MAX register into the CNT_CNTR register to create a new boundary limit.
2 (R0/W)	W1LCNTMIN	Write 1 CNTR Load from MIN. The CNT_CMD.W1LCNTMIN bit loads the current value in the CNT_MIN register into the CNT_CNTR register to create a new boundary limit.
0 (R0/W)	W1LCNTZERO	Write 1 CNTR to Zero. Writing a 1 to the CNT_CMD.W1LCNTZERO bit clears the CNT_CNTR register.

Counter Register

The `CNT_CNTR` register holds the 32-bit, two's-complement count value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows use of the CNT as a 16-bit counter if sufficient for the application.

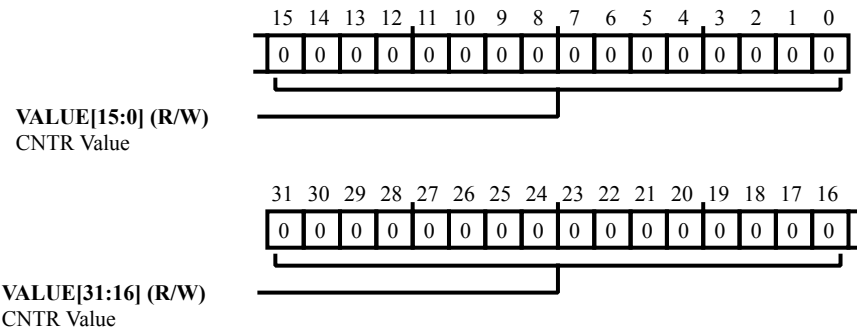


Figure 19-6: CNT_CNTR Register Diagram

Table 19-10: CNT_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CNTR Value. The <code>CNT_CNTR.VALUE</code> bit field holds the 32-bit, two's-complement count value.

Debounce Register

The `CNT_DEBNCE` register selects the noise filtering characteristic of the three input pins according to the formula:

$$t_{\text{filter}} = 128 \times (2^{\text{DPRESCALE}} / \text{SCLK})$$

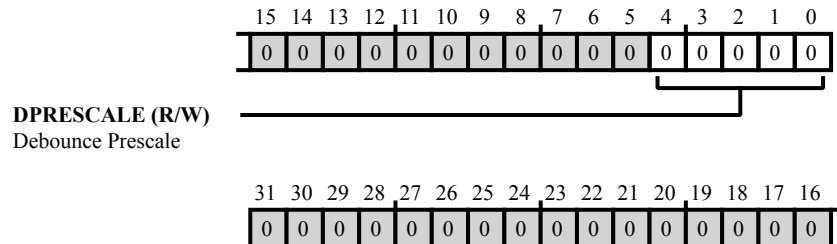


Figure 19-7: CNT_DEBNCE Register Diagram

Table 19-11: CNT_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	DPRESCALE	Debounce Prescale. The <code>CNT_DEBNCE.DPRESCALE</code> selects the desired number of input filtering cycles (and resulting input debounce time) in multiples of SCLK.
		0 1x cycles = 128 SCLK cycles
		1 2x cycles
		2 4x cycles
		3 8x cycles
		4 16x cycles
		5 32x cycles
		6 64x cycles
		7 128x cycles
		8 256x cycles
		9 512x cycles
		10 1024x cycles
		11 2048x cycles
		12 4096x cycles
		13 8192x cycles
		14 16384x cycles
		15 32768x cycles
		16 65536x cycles

Table 19-11: CNT_DEBNCE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		17	131072x cycles
		18	Reserved from this value. The values 10010 - 11111 are reserved.
		31	Reserved until this value

Interrupt Mask Register

The `CNT_IMSK` register supports enabling (unmasking) interrupt request generation from each of the CNT events.

All bits in `CNT_IMSK` either disable/mask an interrupt request (if bit cleared) or enable/unmask an interrupt request (if bit set).

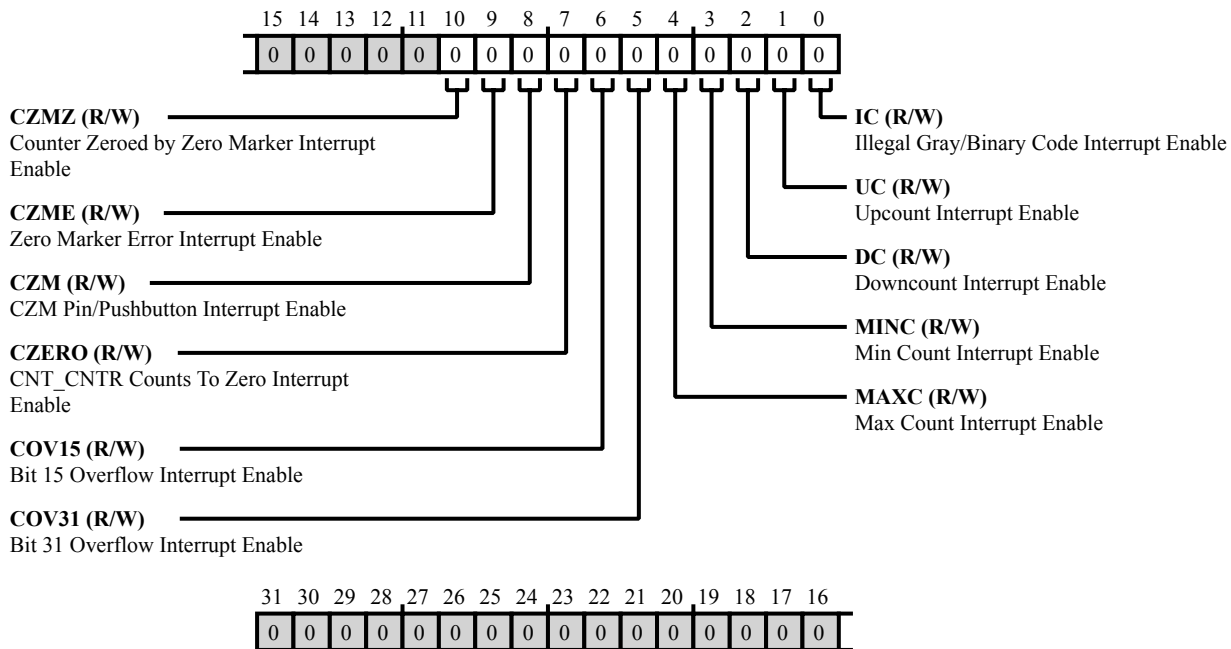


Figure 19-8: `CNT_IMSK` Register Diagram

Table 19-12: `CNT_IMSK` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	CZMZ	Counter Zeroed by Zero Marker Interrupt Enable. The <code>CNT_IMSK.CZMZ</code> bit enables (unmasks) the counter zeroed by zero marker interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
9 (R/W)	CZME	Zero Marker Error Interrupt Enable. The <code>CNT_IMSK.CZME</code> bit enables (unmasks) the zero marker error interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt

Table 19-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	CZM	CZM Pin/Pushbutton Interrupt Enable. The CNT_IMSK.CZM bit enables (unmasks) the CZM pin/pushbutton interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
7 (R/W)	CZERO	CNT_CNTR Counts To Zero Interrupt Enable. The CNT_IMSK.CZERO bit enables (unmasks) the counts to zero interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
6 (R/W)	COV15	Bit 15 Overflow Interrupt Enable. The CNT_IMSK.COV15 bit enables (unmasks) the bit 15 overflow interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable. The CNT_IMSK.COV31 bit enables (unmasks) the bit 31 overflow interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
4 (R/W)	MAXC	Max Count Interrupt Enable. The CNT_IMSK.MAXC bit enables (unmasks) the max count interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
3 (R/W)	MINC	Min Count Interrupt Enable. The CNT_IMSK.MINC bit enables (unmasks) the min count interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt Enable. The CNT_IMSK.DC bit enables (unmasks) the down count interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt

Table 19-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	UC	Upcount Interrupt Enable. The CNT_IMSK.UC bit enables (unmasks) the up count interrupt request.
		0 Mask Interrupt
		1 Unmask Interrupt
0 (R/W)	IC	Illegal Gray/Binary Code Interrupt Enable. The CNT_IMSK.IC bit enables (unmasks) the illegal Gray/Binary Code interrupt request and should only be used in these modes.
		0 Mask Interrupt
		1 Unmask Interrupt

Maximum Count Register

The `CNT_MAX` register holds the 32-bit, two's-complement, higher boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

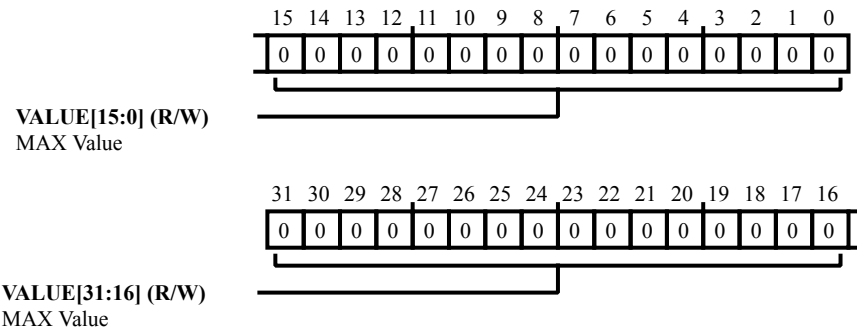


Figure 19-9: CNT_MAX Register Diagram

Table 19-13: CNT_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MAX Value. The <code>CNT_MAX.VALUE</code> bit field holds the 32-bit, two's-complement, higher boundary value.

Minimum Count Register

The `CNT_MIN` register holds the 32-bit, two's-complement, lower boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

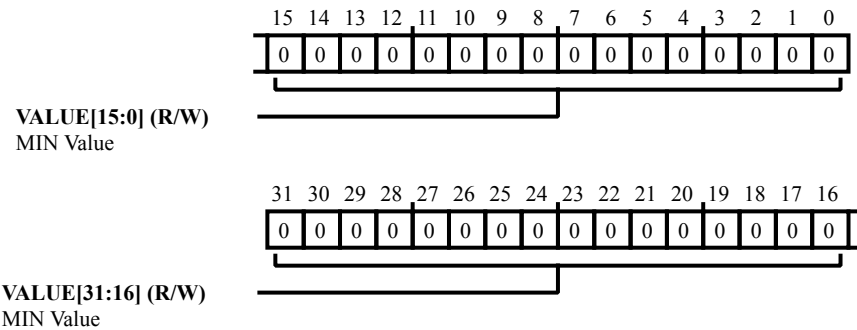


Figure 19-10: `CNT_MIN` Register Diagram

Table 19-14: `CNT_MIN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MIN Value. The <code>CNT_MIN.VALUE</code> bit field holds the 32-bit, two's-complement, lower boundary value.

Status Register

The `CNT_STAT` register provides status information for each of the CNT events as configured in the `CNT_IMSK` register. When a CNT event is detected, the corresponding bit in this register is set. It remains set until either software writes a 1 to the bit (write-1-to-clear) or the CNT is disabled.

All bits in the `CNT_STAT` register indicate either no interrupt request pending (if bit cleared) or an interrupt request pending (if bit set).

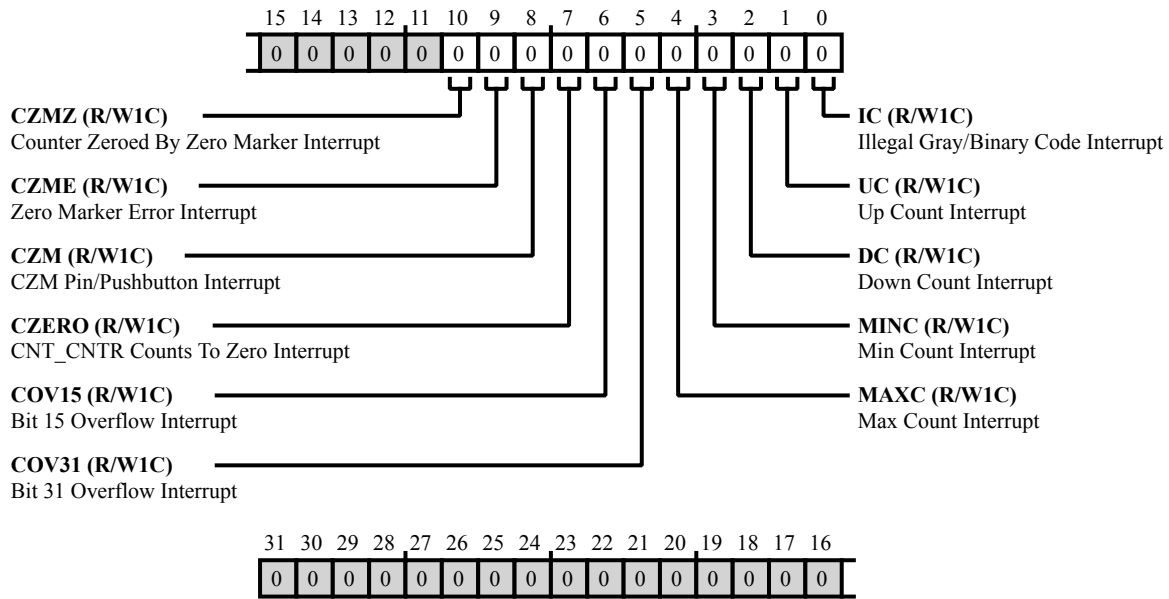


Figure 19-11: CNT_STAT Register Diagram

Table 19-15: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W1C)	CZMZ	Counter Zeroed By Zero Marker Interrupt. The <code>CNT_STAT.CZMZ</code> bit indicates a zero marker error. If the <code>CNT_CFG.ZMZC</code> bit =1, this interrupt request is generated when the <code>CZMII</code> latch reports a significant edge on the <code>CZM</code> input. Once cleared by software the <code>CNT_STAT.CZM</code> bit is not set again when the <code>CZM</code> input remains active without pulsing.
		0 No error
		1 Error occurred

Table 19-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W1C)	CZME	Zero Marker Error Interrupt. The CNT_STAT.CZME bit behaves similarly to the CNT_STAT.CZM bit, with the exception that CNT_STAT.CZME is not set on the CZM edge when the lower four bits of the CNT_CNTR are not zero. In many applications this indicates an error condition, as the zero marker might be out of sync with the counter.
		0 No error
		1 Error occurred
8 (R/W1C)	CZM	CZM Pin/Pushbutton Interrupt. The CNT_STAT.CZM bit indicates a CZM pin/pushbutton error. This interrupt request is generated when a significant edge is seen on the CZM pin, regardless what mode the counter is operating in. This is often used to sense push buttons (especially with the debouncing circuit enabled).
		0 No error
		1 Error occurred
7 (R/W1C)	CZERO	CNT_CNTR Counts To Zero Interrupt. The CNT_STAT.CZERO bit indicates a counts to zero error. This error is generated when the CNT_CNTR register has incremented or decremented toward 0x0000.0000. The latch is not set when software writes to the CNT_CNTR register directly or when the counter is zeroed by writes to the CNT_CMD register.
		0 No error
		1 Error occurred
6 (R/W1C)	COV15	Bit 15 Overflow Interrupt. The CNT_STAT.COV15 bit indicates a bit 15 overflow error. This error is generated when the 16-bit twos-complement CNT_CNTR register has incremented from 0xxxxx.7FFF to 0xxxxx.8000 or decremented from 0xxxxx.8000 to 0xxxxx.7FFF.
		0 No error
		1 Error occurred
5 (R/W1C)	COV31	Bit 31 Overflow Interrupt. The CNT_STAT.COV31 bit indicates a bit 31 overflow error. This error is generated when the 32-bit twos-complement CNT_CNTR register has incremented from 0x7FFF.FFFF to 0x8000.0000 or decremented from 0x8000.0000 to 0x7FFF.FFFF.
		0 No error
		1 Error occurred

Table 19-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	MAXC	Max Count Interrupt. The CNT_STAT.MAXC bit indicates a max count error. This interrupt is used in boundary compare (BND_COMP) mode. If after incrementing the CNT_CNTR register equals CNT_MAX, the CNT_STAT.MAXC bit is set.
		0 No error
		1 Error occurred
3 (R/W1C)	MINC	Min Count Interrupt. The CNT_STAT.MINC bit indicates a minimum count error. This interrupt is used in boundary compare (BND_COMP) mode. If, after decrementing, the CNT_CNTR register equals CNT_MIN, the CNT_STAT.MINC bit is set.
		0 No error
		1 Error occurred
2 (R/W1C)	DC	Down Count Interrupt. The CNT_STAT.DC bit indicates a down count error. This interrupt is generated when the CNT_CNTR register decrements.
		0 No error
		1 Error occurred
1 (R/W1C)	UC	Up Count Interrupt. The CNT_STAT.UC bit indicates an up count interrupt. This interrupt is generated when the CNT_CNTR register increments.
0 (R/W1C)	IC	Illegal Gray/Binary Code Interrupt. The CNT_STAT.IC bit indicates a illegal Gray/Binary Code interrupt and should only be used in these modes. In normal operation those codes can increment or decrement the CNT_CNTR register by one at a time. If the sensed inputs instruct the counter to increment or decrement by two, the CNT_STAT.IC bit is set. Hardware sets the CNT_STAT.IC bit in QUAD_ENC and BIN_ENC encoder modes only.
		0 No error
		1 Error occurred

20 Media Local Bus (MLB)

Media Local Bus (MediaLB[®]) is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The MLB topology supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

The MLB module serves as an interface between the MediaLB and the processor, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. It supports up to 64 logical channels with up to 128 bytes of data per MediaLB frame. Transmit and receive data can be transferred between MediaLB and on-chip memory with DMA block transfers.

The MLB supports the MOST25, MOST50 and MOST150 standards and this document assumes familiarity with these standards. For more information, refer to the Media Local Bus specification version 4.2.

- Copyright 1998-2016 Microchip Technology Inc. All rights reserved. Portions of this chapter are included with permission from Microchip Technology, Inc.

Features

The objective of MLB is to map all the MOST Network data types (transport methods) into a single low-cost, scalable, and standardized hardware interface between a MediaLB controller and at least one other MediaLB Device. The adoption of MediaLB simplifies the hardware interface, reduces the pin count, and facilitates the design of modular reusable hardware. From a software development perspective, the use of MediaLB relieves the system developer from the complexity of the MOST network, which simplifies software development and enables the design of reusable software for different applications. This simplified, standardized interface shortens time to market and makes software maintenance effortless.

- Compliant to Media Local Bus specification version 4.2
- Support MOST25, MOST50, MOST150 standards
- 3-pin mode support various data rates of $256 \times FS$, $512 \times FS$, $1024 \times FS$
- Support 64 logic channels
- Shared pins for 3-pin mode

- Dedicated internal RAM for data buffering and channel table
- MLB 3-pin clock available out on a shared pin for DAI

MLB Definitions

The following are standard MLB and MOST terms that are used in this chapter.

AGU

Address Generation Units. To access a particular HBI channel the HC must first configure one of two HBI address generation units. AGUs can be configured by writing the HBI command registers, HCMD0 and HCMD1.

CAT

Channel Allocation Table. The Channel Allocation Table (CAT) is comprised of 16 CTR entries. Each 16-bit CAT entry represents a logical connection to or from a transmit/receive device (for example MediaLB or HBI channel).

CDT

Channel Descriptor Table.

FCE

Flow Control Enable bit. The FCE bit is used by MediaLB isochronous Rx channels only.

HBI

Host Bus Interface. The HBI block provides 16-bit parallel slave access to all MOST channels and data types for the external Host Controller (HC). The HBI supports up to 64 independent channels.

HC

Host Controller (external).

HCMDx

HBI Command registers.

HSTSx

HBI status registers.

MFE

Multi-Frame per sub-buffer enable bit. The MFE bit is used by MediaLB synchronous channels only.

PML

Packet Message Length.

Clocking

The MLB controller provides an external clock pin—the media local bus clock. The MLB controller generates the clock. It is synchronized to the MOST network and provides the timing for the entire MLB interface at FS = 48 kHz.

Functional Description

The *MediaLB Block Diagram* figure shows the MLB high-level architecture. The MLB core serves as an interface between the MediaLB and the processor, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. The MLB core has the following responsibilities.

- Transmit commands and data when functioning as the transmitting device associated with a *Channel Address*
- Receive data and transmit Rx status responses when functioning as the receiving device associated *Channel Address*
- MLB lock detection
- System channel command handling

The MediaLB interfaces are capable of exchanging data at speeds up to $1024 \times F_s$ in 3-pin mode.

A set of physical channels for exchanging data over the MediaLB bus is supported. These physical channels (4 bytes in length, or a quadlet) can be grouped into logical channels, where each logical channel is referenced using a channel address and represents a uni-directional datapath between a specific MediaLB device transmitting the data and the MediaLB device(s) receiving the data. The MediaLB 3-pin interface provides support for up to 128 bytes of data per frame. The logical channels, configured by system software, can be any combination of channel types (synchronous, asynchronous, isochronous, or control) and direction (transmit or receive).

ADSP-2156x MLB Register List

The MediaLB Device Interface Macro 2 (MediaLB DIM 2), also referred to as OS62420, implements the required functionality of a Media Local Bus (MediaLB) device. This logic serves as an interface between the inter-chip MediaLB bus and a customer IC, implementing the physical- and link-layer requirements outlined in the MediaLB Specification.

Table 20-1: ADSP-2156x MLB Register List

Name	Description
MLB_ACMR0	Peripheral Channel Mask 0 Register

Table 20-1: ADSP-2156x MLB Register List (Continued)

Name	Description
MLB_ACMR1	Peripheral Channel Mask 1 Register
MLB_ACSR0	Peripheral Channel Status 0 Register
MLB_ACSR1	Peripheral Channel Status 1 Register
MLB_ACTL	Bus Control Register
MLB_CTL0	MediaLB Control 0 Register
MLB_CTL1	Control 1 Register
MLB_GCTL	MLB Global Control Register
MLB_HCBR0	HBI Channel Busy 0 Register
MLB_HCBR1	HBI Channel Busy 1 Register
MLB_HCER0	HBI Channel Error 0 Register
MLB_HCER1	HBI Channel Error 1 Register
MLB_HCMR0	HBI Channel Mask 0 Register
MLB_HCMR1	HBI Channel Mask 1 Register
MLB_HCTL	HBI Control Register
MLB_MADR	Memory Interface Address Register
MLB_MCTL	Memory Interface Control Register
MLB_MDAT0	Memory Interface Control Data 0 Register
MLB_MDAT1	Memory Interface Control Data 1 Register
MLB_MDAT2	Memory Interface Control Data 2 Register
MLB_MDAT3	Memory Interface Control Data 3 Register
MLB_MDWE0	Memory Interface Control Data Write Enable 0 Register
MLB_MDWE1	Memory Interface Control Data Write Enable 1 Register
MLB_MDWE2	Memory Interface Control Data Write Enable 2 Register
MLB_MDWE3	Memory Interface Control Data Write Enable 3 Register
MLB_MIEN	Interrupt Enable Register
MLB_MS0	Channel Status 0 Register
MLB_MS1	Channel Status 1 Register
MLB_MSD	System Data Register
MLB_MSS	System Status Register

ADSP-2156x MLB Interrupt List

Table 20-2: ADSP-2156x MLB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
123	MLB0_INT0	MLB0 Interrupt 0		
124	MLB0_INT1	MLB0 Interrupt 1		
125	MLB0_STAT	MLB0 Status		

MediaLB Protocol

The MediaLB topology supports communication among all MediaLB devices, including the MediaLB controller. The bus interface consists of a uni-directional line for clock (MLBC), a bidirectional line for signal information (MLBS), and a bidirectional line for data transfer (MLBD). The MediaLB topology supports one controller connected to one or more devices, where the controller is the interface between the MediaLB devices and the MOST network.

The MediaLB controller includes MediaLB device functionality, and also generates the MediaLB clock (MLBC) that is synchronized to the MOST Network. This generated clock provides the timing for the entire MediaLB interface. The MLBS line is a multiplexed signal which carries channel addresses generated by the MediaLB controller, as well as command and RxStatus bytes from MediaLB devices. The MLBD line is driven by the transmitting MediaLB device and is received by all other MediaLB devices, including the MediaLB controller. The MLBD line carries the actual data (synchronous, asynchronous, control, or isochronous).

Once per MOST network frame, the MLB controller generates a unique frame sync pattern on the MLB_SIG line. The end of the frame sync pattern defines the byte boundary and the channel boundary for the MLB_SIG and MLB_DAT lines of all MLB devices.

The MLB controller manages the arbitration for all the channels on the MLB and grants bandwidth for all the MLB devices. An MLB *physical channel* is defined as four bytes wide, or a quadlet. Physical channels can be grouped into multiple quadlets (which do not have to be consecutive) to form an MLB *logical channel*, which is defined by a unique channel address.

As shown in *MLB Data Structure*, the MLB controller initiates communication by sending out a channel address on the MLB_SIG line for each physical channel. The channel address indicates which MLB device is transmitting and which MLB devices are receiving in the following physical channel. Therefore, four bytes after the controller outputs the channel address on the MLB_SIG line, the transmitting device outputs a command byte on the MLB_SIG line and outputs the respective data on the MLB_DAT line, concurrently. The MLB command byte contains the type of data currently being transmitted (for example synchronous, asynchronous or control).

The MLB device receiving the channel data outputs a status byte, RxStatus, on the MLB_SIG line immediately after the transmitting device outputs the command byte. The status response can indicate that the receiving device is busy and cannot receive the data at present, or the device is ready to receive the data. Since synchronous stream data is sent in a broadcast fashion, receiving devices cannot return a busy status and should not drive RxStatus onto the MLB_SIG line.

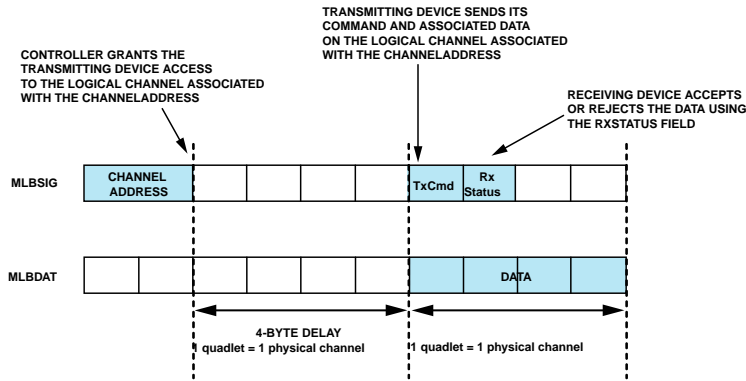


Figure 20-1: MLB Data Structure

MLB Architectural Concepts

The following sections provide information about the MLB architecture.

MediaLB Block Diagram

The *MediaLB Block Diagram* shows the various blocks within the interface its connections to the processor.

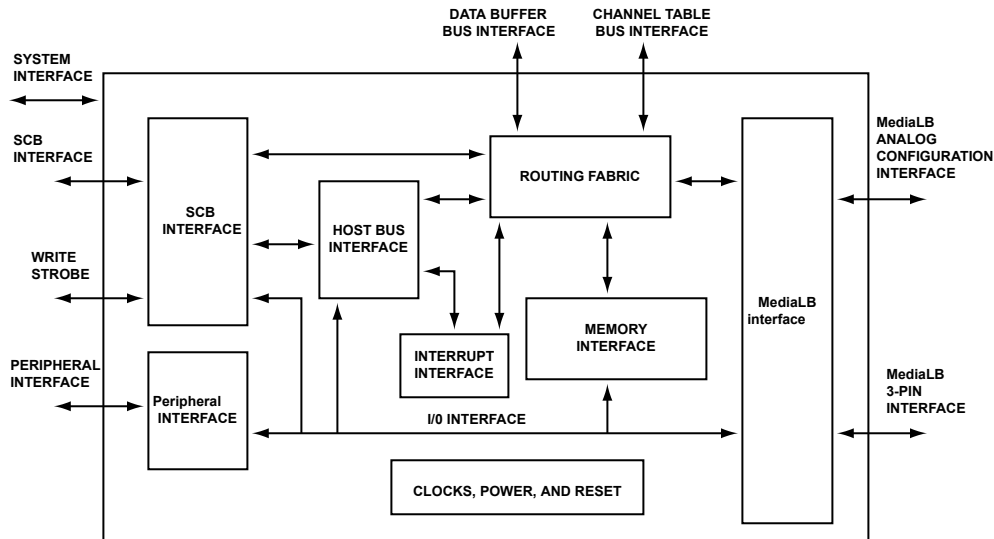


Figure 20-2: MediaLB Block Diagram

MediaLB Interface

The MediaLB interfaces provide real-time access to all network data types - synchronous, asynchronous, control, and isochronous data.

- MediaLB 3-pin interface – Supports the MediaLB protocol for single-ended 3-pin mode, with a maximum data rate of 1024 FS (49.152 MHz at FS = 48 kHz).

MediaLB Channel Address to Logical Channel Mapping

The MediaLB channel addresses are mapped to the logical channels as shown in the *MediaLB Channel Address to Logical Channel Mapping* table.

Table 20-3: MediaLB Channel Address to Logical Channel Mapping

Channel Address	Logical Address
0x0002	1
0x0004	2
0x0006	3
...	...
0x007C	62
0x007E	63
0x01FE	0 (Logical channel 0 is the system channel and is reserved)

Routing Fabric

The Routing Fabric (RF) block manages the flow of data between the MediaLB port and the HBI port. It manages accessing the Channel Table RAM (CTR) and Data Buffer RAM (DBR), which are explained in the following subsections. The routing fabric uses channel descriptors (stored in the CTR) to manage access to dynamic buffers in the DBR.

Data Buffer RAM

The Data Buffer RAM (DBR) is an 8-bit x 16k entries deep, single-port synchronous SRAM and provides dynamic circular buffering between the transmit and receive devices. The size and location of each data buffer is defined by software in the Channel Descriptor Table (CDT), which is located in the Channel Table RAM (CTR), described in following sections.

Channel Table RAM

The Channel Table RAM (CTR) is a 128-bit x 144-entry table that allows system software to dynamically configure channel routing and allocate data buffers in the DBR. The CTR is logically divided into three tables:

- [Channel Descriptor Tables](#)
- [AHB Descriptor Table \(ADT\)](#)
- [Channel Allocation Table](#)

Address Mapping

The *CTR Address Mapping* table shows the address mapping for the CTR.

Table 20-4: CTR Address Mapping

Label	Address	Bits [127:96]	Bits [95:64]	Bits [63:32]	Bits [31:0]				
Channel Descriptor Table (CDT)									
CDT	0x00	CDT0[127:0], CL = 0							
	0x01	CDT1[127:0], CL = 1							
	0x02	CDT2[127:0], CL = 2							
	...								
	0x3D	CDT61[127:0], CL = 61							
	0x3E	CDT62[127:0], CL = 62							
	0x3F	CDT63[127:0], CL = 63							
AHB Descriptor Table (ADT)									
ADT	0x40	ADT0[127:0], CL = 0							
	0x41	ADT1[127:0], CL = 1							
	0x42	ADT2[127:0], CL = 2							
	...								
	0x7D	ADT61[127:0], CL = 61							
	0x7E	ADT62[127:0], CL = 62							
	0x7F	ADT63[127:0], CL = 63							
Channel Allocation Table (CAT)									
CAT for MediaLB	0x80	CAT7	CAT6	CAT5	CAT4	CAT3	CAT2	CAT1	CAT0

	0x87	CAT63	CAT62	CAT61	CAT60	CAT59	CAT58	CAT57	CAT56
CAT for HBI*1	0x88	CAT71	CAT70	CAT69	CAT68	CAT67	CAT66	CAT65	CAT64

	0x8F	CAT127	CAT126	CAT125	CAT124	CAT123	CAT122	CAT121	CAT120

*1 A fixed relationship exists between ADT entries and HBI CAT entries. When using HBI channel 0 (CAT64), program ADT0. When using HBI channel 1 (CAT65), program ADT1, and so on.

Channel Allocation Table

The *Channel Allocation Table (CAT)* table is comprised of 16 CTR entries (addresses 0x80-0x8F) as shown in the *CTR Entry Map* table. Each 16-bit CAT entry represents a logical connection to or from a transmit or receive device. (for example, MediaLB channel). All entries are indexed according to a fixed physical address assigned to every RX/TX channel as shown in the *CAT Entry Formats* table. The value stored in a CAT entry includes a 6-bit connection label, which provides a pointer to the CDT. To complete a logical channel and form a routing connection, system software must assign the same connection label to both the RX and TX channels.

Table 20-5: CAT Entry Map

Peripheral	TX Channels	RX Channels	CAT Start Index	CAT End Index	Entries
MediaLB	0 to 64	64 TX Channels	0	63	64
HBI	0 to 64	64 TX Channels	64	127	64

The format of a full CAT entry is shown in the *CAT Entry Formats* table, with field descriptions described in the *CAT Field Definitions* table. All reserved bits of a CAT entry field should be written as zero.

Table 20-6: CAT Entry Formats

Channel Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Isochronous	rsvd	FCE	rsvd	RNW	CE	CT[2:0] = 011			rsvd	CL[5:0]						
Asynchronous	rsvd		MT	RNW	CE	CT[2:0] = 010			rsvd	CL[5:0]						
Control	rsvd		MT	RNW	CE	CT[2:0] = 001			rsvd	CL[5:0]						
Synchronous	rsvd	MFE	MT	RNW	CE	CT[2:0] = 000			rsvd	CL[5:0]						

Table 20-7: CAT Field Definitions

Field	Description
CL[5:0]	Connection Label (offset into CDT)
CT[2:0]	Channel Type (others) 111 = Reserved 110 = Reserved 101 = Reserved 100 = Reserved 011 = Isochronous 010 = Asynchronous 001 = Control 000 = Synchronous
CE	Channel Enable. 0 = Disabled, 1 = Enabled
RNW	Read Not Write. 0 = Write, 1 = Read
MT	Mute Enable. 0 = Disabled When set for synchronous channels, the MT bit forces RX channels to write zeros into the channel data buffer, and TX channels to output zeros on the physical interface. When set for asynchronous and control channels, the MT bit causes DMA to halt at a packet boundary. Not valid for isochronous channels.
FCE	Flow Control Enable. 0 = Disabled, 1 = Enabled The FCE bit is used by MediaLB isochronous RX channels only.
MFE	Multi-Frame per sub-buffer enable. 0 = Disabled, 1 = Enabled The MFE bit is used by MediaLB synchronous channels only.

Table 20-7: CAT Field Definitions (Continued)

Field	Description
rsvd	Reserved. Software writes a 0 to all reserved bits when the entry is initialized. These bits are read-only after initialization.

Channel Set Up

Data direction is in reference to the DBR. The data direction of CAT entries corresponding to the same channel is reversed for the HBI CAT and the MediaLB CAT.

- For a Tx channel (from the HC to the MediaLB interface):
 - HBI CAT entry: RNW = 0 (write)
 - MediaLB CAT entry: RNW = 1 (read)
- Conversely, for an Rx channel (data from MediaLB to HC):
 - HBI CAT entry: RNW = 1 (read)
 - MediaLB CAT entry: RNW = 0 (write)

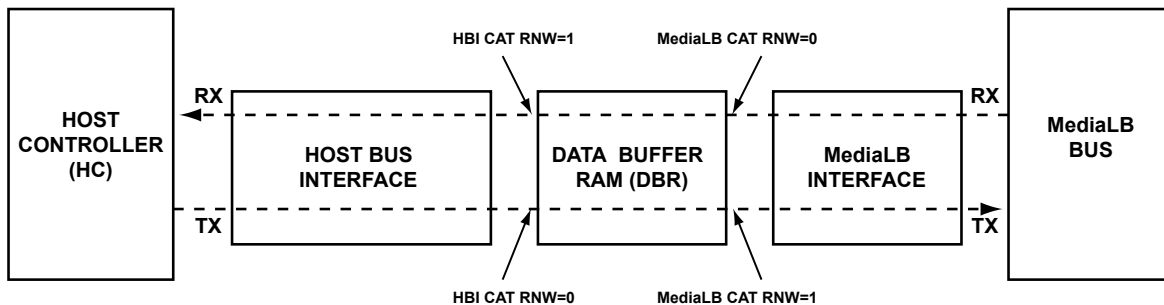


Figure 20-3: DBR Directional Relationship

Channel Descriptor Tables

The Channel Descriptor Table (CDT) is comprised of 64 CTR entries (addresses 0x00 - 0x3F), as shown in the [Table 20-4 CTR Address Mapping](#) table. Each 128-bit CDT entry (also referred to as a channel descriptor) is referenced by a connection label and contains information about a data buffer in the DBR (for example buffer size, address pointers). The format of each CDT entry is dependent on the channel type (synchronous, isochronous, asynchronous, or control).

NOTE: All reserved channel descriptor bits must be written to 0 by software when initialized.

Synchronous Channel Descriptors

The format and field definitions for a synchronous CDT entry are shown in the *Synchronous CDT Entry Format* and *Synchronous CDT Entry Field Definitions* tables.

Table 20-8: Synchronous CDT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WSBC[1:0]		Reserved													
16	RSBC[1:0]		Reserved													
32	Reserved															
48	Reserved															
64	WSTS[3:0]				WPTR[11:0]											
80	RSTS[3:0]				RPTR[11:0]											
96	Reserved				BD[11:0]											
112	Reserved		BA[13:0]													

Table 20-9: Synchronous CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. BD = 4 m bpf 1 where: m = frames per sub-buffer (for MFE = 0, m = 1) bpf = bytes per frame.	RW
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA read address = BA + WPTR.	RWU
RSBC	Read Sub-buffer Counter	Software initializes to 0, hardware updates. Counts the read sub-buffer offset. DMA uses for pointer management.	RWU
WSBC	Write Sub-buffer Counter	Software initializes to 0, hardware updates. Counts the write sub-buffer offset. DMA uses for pointer management.	RWU
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: xxx0 = normal operation (no mute) xxx1 = normal operation (mute) xx0x = idle	RWU

Table 20-9: Synchronous CDT Entry Field Definitions (Continued)

Field	Description	Details	Access
WSTS	Write Status	Software initializes to 0, hardware updates. WSTS States: xxx0 = normal operation (no mute) xxx1 = normal operation (mute) xx0x = idle 1xxx = command protocol error	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

Isochronous Channel Descriptors

The format and field definitions for a synchronous CDT entry are shown in the *Isochronous Entry Formats* and *Isochronous CDT Entry Field Definitions* tables.

Table 20-10: Isochronous Entry Formats

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Reserved															
16	Reserved															
32	Reserved							BS[8:0]								
48	Reserved															
64	WSTS[3:0]				WPTR[12:0]											
80	RSTS[3:0]				RPTR[12:0]											
96	Reserved				BD[12:0]											
112	BF	Rsvd	BA[13:0]													

Table 20-11: Isochronous CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR.	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. Isochronous buffers must be large enough to hold at least 3 blocks (packets) of data. BD Must be an integer multiple of blocks.	RW

Table 20-11: Isochronous CDT Entry Field Definitions (Continued)

Field	Description	Details	Access
BF	Buffer Full	Software initializes to 0, hardware updates. DMA write hardware sets BF when the buffer is full. DMA read hardware clears BF when the buffer is empty. BF is valid only when buffer is full or empty, otherwise ignore.	RWU
BS	Block Size	BS defines when to begin the DMA to the data buffer. BS = buffer block size in bytes 1. For RX channels, the DMA writes start when the number of empty bytes in the data buffer the block size. For TX channels, the DMA reads start when the number of valid bytes in the data buffer the block size.	RWU
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA write address = BA + WPTR.	RWU
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: xx1 = active xx0 = idle	RWU
WSTS	Write Status	Software initializes to 0, hardware updates. WSTS States: xxx0 = active xxx1 = idle xx0x = command protocol error 1xxx = buffer overflow (FCE = 0 only)	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

Asynchronous/Control Channel Descriptors

The format and field definitions for an Asynchronous/Control CDT entry are shown in the *Asynchronous/Control CDT Entry Format* and *Asynchronous/Control CDT Entry Field Definitions* tables.

Table 20-12: Asynchronous/Control CDT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WPC[4:0]					Reserved										
16	RPC[4:0]					Reserved										
32	Rsvd	WPC[7:5]			Reserved											
48	Rsvd	RPC[7:5]			Reserved											
64	WSTS[3:0]				WPTR[11:0]											
80	RSTS[3:0]				RPTR[11:0]											
96	RSTS[4]	WSTS[4]	Rsvd		BD[11:0]											
112	Rsvd		BA[13:0]													

Table 20-13: Asynchronous/Control CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR.	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. BD = max packet length 1.	RW
RPC	Read Packet Count	Software initializes to 0, hardware updates. Used with RPC, RPTR and WPTR to determine if the buffer is empty or full.	RWU
WPC	Write Packet Count	Software initializes to 0, hardware updates. Used with RPC, RPTR and WPTR to determine if the buffer is empty or full.	RWU
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA read address = BA + WPTR.	RWU
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: x0x00 = idle xx1xx = <i>ReceiverProtocolError</i> response received from RX device 1xxxx = <i>ReceiverBreak</i> command received from RX device	RWU

Table 20-13: Asynchronous/Control CDT Entry Field Definitions (Continued)

Field	Description	Details	Access
WSTS	Write Status	Software initializes to 0, hardware updates. Status States (only valid for DMA pointers associated with the MLB block, not HBI block pointers): xxx0 = idle xxx1 = command protocol error detected xx0x = <i>AsyncBreak/ControlBreak</i> command received from TX device	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

AHB Descriptor Table (ADT)

The AHB block manages data exchange between local channel data buffers within MLB module and the system memory buffer. To support system memory buffering, a ping-pong memory structure is implemented on a per channel basis using 128-bit descriptors for AHB Descriptor Table (ADT) entries. The [Table 20-4 CTR Address Mapping](#) table provides a complete address map of the CTR, including the location of the ADT.

Each logical channel is assigned a separate 128-bit descriptor, defining the data buffers in the system memory used by the DMA interface for that channel. The descriptors are stored at fixed addresses in the CTR as described in previous section. The *ADT Field Definitions* table provides an overview of field definitions for ADT entries.

Table 20-14: ADT Field Definitions

Field	No. of Bits	Description	Access
CE	1	Channel Enable. 0 = Disabled	RW, U
LE	1	Endianness Select. 0 = Big Endian, 1 = Little Endian	RW
PG	1	Page pointer. Software initializes to 0, hardware writes thereafter. 0 = Ping buffer, 1 = Pong buffer	RW, U
RDY1	1	Buffer Ready bit for ping buffer page. 0 = Not ready, 1 = Ready	RW
RDY2	1	Buffer Ready bit for pong buffer page. 0 = Not ready, 1 = Ready	RW
DNE1	1	Buffer Done bit for ping buffer page. 0 = Not done, 1 = Done	R, U, c0
DNE2	1	Buffer Done bit for pong buffer page. 0 = Not done, 1 = Done	R, U, c0
ERR1	1	Error Response detected for ping buffer page. 0 = No error, 1 = Error	R, U, c0
ERR2	1	Error Response detected for pong buffer page. 0 = No error, 1 = Error	R, U, c0
PS1	1	Packet Start bit for ping buffer page. 0 = No packet start, 1 = Packet start Reserved for synchronous and isochronous channels.	RW, U both TX and RX
PS2	1	Packet Start bit for pong buffer page. 0 = No packet start, 1 = Packet start Reserved for synchronous and isochronous channels.	RW, U both TX and RX

Table 20-14: ADT Field Definitions (Continued)

Field	No. of Bits	Description	Access
MEP1	1	Most Ethernet Packet indicator for ping buffer page. 0 = Not MEP, 1 = MEP. MEP1 only valid for the first page of a segmented buffer. Reserved for control synchronous and isochronous channels.	Rsvd for TX, R, U, c0 for RX
MEP2	1	Most Ethernet Packet indicator for pong buffer page. 0 = Not MEP, 1 = MEP. MEP2 only valid for the first page of a segmented buffer. Reserved for control synchronous and isochronous channels.	Rsvd for TX, R, U, c0 for RX
BD1	11 to 13	Buffer Depth for ping buffer page. 11 or 12 bits for asynchronous and control channels. 13 bits for synchronous and isochronous channels.	RW
BD2	11 to 13	Buffer Depth for pong buffer page. 11 or 12 bits for asynchronous and control channels. 13 bits for synchronous and isochronous channels.	RW
BA1	32	Buffer Base Address for ping buffer page	RW
BA2	32	Buffer Base Address for pong buffer page.	RW
Reserved	varies	Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.	RW, U

The *Ping-Pong System Memory Structure* figure shows that this system memory structure is similar for all channel types and shows the relationship between the BAn, BDn, and PG descriptor fields.

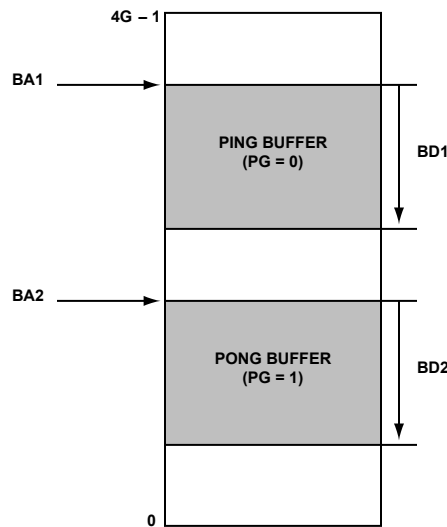


Figure 20-4: Ping-Pong System Memory Structure

Each ADT entry (also referred to as a *Channel Descriptor*) holds a 32-bit BAn field which defines the start of each ping or pong buffer within system memory. The BDn field is used to indicate the size for the respective ping or pong page. The maximum size is 2k entries for asynchronous and control channels and 8k entries for isochronous and synchronous channels.

Synchronous Channel Descriptors

The synchronous buffering scheme allows each ping or pong buffer to contain a single frame or a multiple number of frames. For this reason, the synchronous buffer depth (BD_n) must be defined in terms of an integer number (n), frames per sub-buffer (m) and bytes per frame (bpf) of data (for example BD_n = n m bpf 1). The *Synchronous ADT Entry Format* table shows the format for a synchronous ADT entry. The field definitions are defined in the *ADT Field Definitions* table. Each synchronous channel buffer can be up to 8k-bytes deep.

Table 20-15: Synchronous ADT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	BD1[12:0]												
48	RDY2	DNE2	ERR2	BD2[12:0]												
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Isochronous Channel Descriptors

The isochronous buffering scheme allows each ping or pong buffer to contain a single block or a multiple number of blocks. For this reason, the isochronous buffer depth (BD_n) must be defined in terms of an integer number (n) and block size (BS) (for example BD_n = n (BS + 1) 1).

The *Isochronous ADT Entry Format* table shows the format for an isochronous ADT entry. The field definitions are defined in the *ADT Field Definitions* table. Each isochronous channel buffer can be up to 8k-bytes deep.

Table 20-16: Isochronous ADT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	BD1[12:0]												
48	RDY2	DNE2	ERR2	BD2[12:0]												
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Asynchronous and Control Channel Descriptors

Every asynchronous and control packet adheres to the Port Message Protocol (PMP), which designates the first two bytes of each packet as the packet length (PML). Each packet must be no more than 2048-bytes (PML 2048).

Software must set the buffer ready bit (RDY_n) for each buffer as it programs the DMA. As hardware processes each buffer, it sets the done bit (DNE_n) and generates an interrupt to inform HC. When hardware finishes processing a buffer, it can begin processing another buffer if RDY_n is set. The application is responsible for setting up and configuring the channel buffer descriptor prior to every DMA access on the channel.

Two packet buffering modes are supported by hardware for programming the DMA, single-packet mode (MLB_ACTL.MPB =0) and multiple-packet mode (MLB_ACTL.MPB =1). The MPB is written prior to enabling the channel DMA.

Single Packet Mode. The single-packet mode asynchronous and control buffering scheme supports a maximum of one packet per buffer (for example, ping or pong). Both non-segmented and segmented data packets are allowed while using single-packet mode. Non-segmented packets are exchanged when only one buffer (for example, ping or pong) is needed for packet transfer. Segmented packets are exchanged when a single packet is too long for one buffer and the packet must span multiple buffers.

The *Single-Packet Asynchronous and Control Entry Format* table shows the format for single-packet mode asynchronous and control ADT entries. The field definitions are defined in the *ADT Field Definitions* table.

Table 20-17: Single-Packet Asynchronous and Control Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	PS1	MEP1	BD1[10:0]										
48	RDY2	DNE2	ERR2	PS2	MEP2	BD2[10:0]										
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Multiple Packet Mode. The multiple-packet mode asynchronous and control buffering scheme supports more than one packet per system memory buffer, as shown in the *Asynchronous/Control CDT Entry Format* table. Multiple-packet mode reduces the interrupt rate for packet channels at the cost of increasing buffering and latency.

For TX packet channels in multiple-packet mode, software sets the packet start bit (PS_n) for every buffer. Setting PS_n informs hardware that the first two bytes of the buffer contains the port message length (PML) of the first packet. After the first packet, hardware keeps track of where packets start and end within the current buffer. Software should not write to PS_n while the buffer is active (RDY_n = 1 and DNE_n = 0). For TX packet channels, the buffer is done (DNE_n = 1) when the last byte of the last packet in the buffer is read from system memory. Software should

set the buffer depth to contain the exact number of complete packets for that buffer. Segmented buffers are not supported for TX packet channels in multiple-packet mode.

NOTE: The PS1 and PS2 bits are only valid for TX channels. Set PS1 and PS2 = 1 at the start of the buffer.

Table 20-18: Multiple-Packet Asynchronous and Control Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	PS1	BD1[11:0]											
48	RDY2	DNE2	ERR2	PS2	BD2[11:0]											
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Interrupt Interface Block

The interrupt interface raises an interrupt when specific changes to HBI channel descriptors occur, including:

- For asynchronous and control read/write channels:
 - a packet is available to read in the channel buffer, or
 - sufficient empty space is available in the channel buffer to accept a requested packet write
- For isochronous read/write channels:
 - the number of valid bytes in the channel buffer exceeds the block size, or
 - the number of empty bytes in the channel buffer exceeds the block size

Operating Modes

The following sections describe the operating modes of the MLB interface. The channel type selection enables the logical channels to operate in synchronous, asynchronous, isochronous, or control channels.

NOTE: The logical channels can be any combination of channel type (for example synchronous, asynchronous, or control) and direction (transmit or receive).

Isochronous Data Exchange

An isochronous HBI channel is initially opened and synchronized with `HCMD0.CMD[2:0] = 010`. For isochronous channels, no further synchronization is required from the HBI perspective; however, an optional resynchronization command is available for HC flexibility. Setting `HCMD0.CMD[2:0] = 011` reinitializes the address pointer within

the data buffer, ensuring that subsequent data exchange with the channel is aligned at an isochronous packet boundary. When the HC must close an isochronous channel before it has read or written an entire data packet, setting HCMD0.CMD[2:0]= 000 reopens the channel without synchronizing the address pointer in the buffer. This action allows reading and writing to continue where the HC previously stopped.

For isochronous data transmission, the *Exchanging Isochronous Data on an HBI Channel* figure shows the flow control.

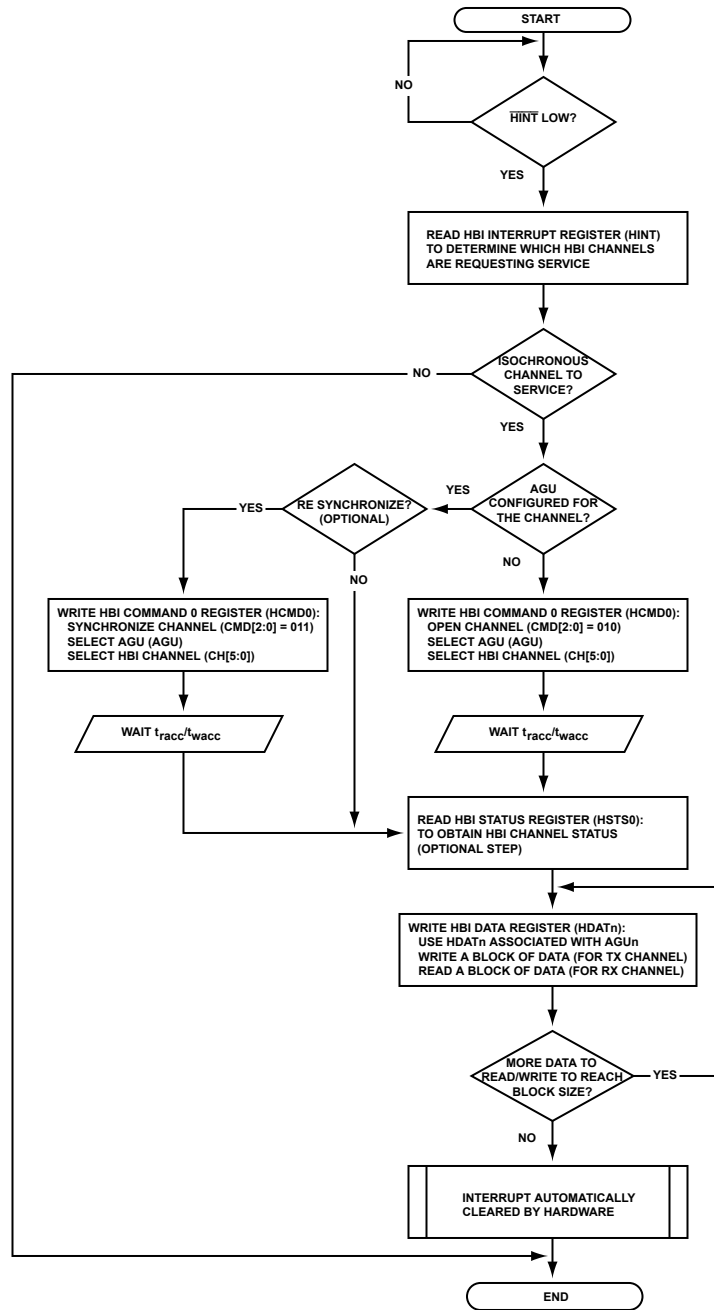


Figure 20-5: Exchanging Isochronous Data on an HBI Channel

Asynchronous and Control Data Exchange

An asynchronous or control HBI channel is initially opened using `HCMD0.CMD[2:0]= 010`. Occasionally, the HC may need to close a packet channel before it has completed writing or reading the current packet of data (for example, the HC needs to use this AGU to service another HBI channel). In this case, the HC can reopen the previous channel with `HCMD0.CMD[2:0]= 000`. This situation allows the HC to continue writing or reading a packet from the point it left off. In this situation, the PML is already known and the packet length is not reread or rewritten by the HC.

During the reading of a packet, if the HC sets `HCMD0.CMD[2:0]= 010` for the channel before the last byte of the packet is read, an internal hardware pointer is reset to the beginning of the packet buffer. This situation requires that the HC reread the PML (from `HSTS1`) and reread the packet data (from `HDATAn`) from the beginning. In the same manner, if the HC resets `HCMD0.CMD[2:0]= 010` for the channel before the last byte of a packet is written, an internal hardware pointer is reset to the beginning of the packet buffer. This situation requires that the HC rewrite the PML (to `HCMD1`) and rewrite the packet data (to `HDATAn`) from the beginning. Any previous packet data in the buffer is overwritten.

Frame synchronization is not supported for asynchronous channels.

Synchronous Data Exchange

The MLB core provides two modes of operation; standard and multi-frame per sub-buffer which provide flexibility for implementing synchronous channels. Channels configured for standard mode require less buffer space, but have higher interrupt rates and more stringent latency requirements. Channels setup for multi-frame per sub-buffer mode require more buffer space, but have lower interrupt rates and less stringent latency requirements.

To set up a channel in multi-frame per sub-buffer mode:

1. Program the `MLB_CTL0.FCNT` bit field to select the number of frames per sub-buffer.
2. Program the `CAT` to enable multi-frame sub-buffering (`MFE= 1`) for each particular channel.
3. Set the buffer depth in the `CDT`: $BD = 4 \times m \times bpf + 1$ where: m = frames per sub-buffer, bpf = bytes per frame.
4. Repeat for additional synchronous channels

A sample synchronous data buffer is shown in the *Synchronous Data Buffer Structure* figure. Each data buffer contains four sub-buffers and each sub-buffer contains space for 1 to 64 frames of data, determined by the `MLB_CTL0.FCNT` bits.

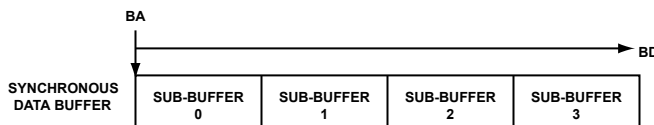


Figure 20-6: Synchronous Data Buffer Structure

Data Transfer

Two modes of operation are supported for transferring channel data between the MLB and internal memory. DMA allows the multi-channel DMA engine to manage data transfers without core intervention. Core driven mode (I/O mode) allows software to manage the transfer of data between MLB and internal memory.

NOTE: All hardware channels must use the same data transfer method. Mixed mode operation where hardware channels operate in both I/O mode and DMA mode is not supported.

DMA

The processor supports DMA mode which uses only INCR8, INCR4 and SINGLE beat bursts. Program the `MLB_ACTL.DMAMODE = 1` to use DMA mode.

Programming Model

This section provides general guidelines for programming the MediaLB interface.

Channel Initialization

The software flow required to initialize a channel must be performed in order to ensure proper operation.

Configure the Hardware

1. Initialize the CTR and registers.
 - a. Set all the CTR (CAT, CDT, and ADT) bits to 0.
 - b. Set all bits of all registers to 0.
2. Configure the MediaLB interface.
 - a. Select 3-pin operation by setting the `MLB_CTL0.PEN = 0`.
 - b. Select MediaLB clock speed via `MLB_CTL0.CLK`.
 - c. Set MediaLB enable via `MLB_CTL0.EN`.
3. Configure the HBI interface.
 - a. Set `MLB_HCMR0` and `MLB_HCMR1 = 0xFFFFFFFF` to activate all channels.
 - b. Set the HBI enable bit: `MLB_HCTL.EN = 1`.

Program the CAT and the CDT

1. Initialize all bits of the CAT to 0.

2. Select a logical channel: $N = 1 - 63$.
3. Program the CDT for channel N .
 - a. Set the 14-bit base address (BA)
 - b. Set the 12-bit or 13-bit buffer depth (BD): $BD = \text{buffer depth in bytes} - 1$
 - For synchronous channels: $(BD + 1) = 4 \text{ frames per sub-buffer (m) bytes-per-frame (bpf)}$
 - For isochronous channels: $(BD + 1) \bmod (BS + 1) = 0$
 - For asynchronous channels: $(BD + 1) \text{ max packet length (1024 for a MOST Data Packet (MDP) 1536 for a MOST Ethernet Packet (MEP))}$
 - For control channels: $(BD + 1) \text{ max packet length (64)}$
4. Program the CAT for the inbound DMA.
 - a. For TX channels (to MediaLB) HBI is the inbound DMA
 - b. For RX channels (from MediaLB) MediaLB is the inbound DMA
 - c. Set the channel direction: $RNW = 0$
 - d. Set the channel type: $CT[2:0] = 010$ (asynchronous), 001 (control), 011 (isochronous), or 000 (synchronous)
 - e. Set the connection label: $CL[5:0] = N$
 - f. If $CT[2:0] = 000$ (synchronous), set the mute bit ($MT = 1$)
 - g. Set the channel enable: $CE = 1$
 - h. Set all other bits of the CAT to 0
5. Program the CAT for the outbound DMA.
 - a. For TX channels (to MediaLB) HBI is the outbound DMA
 - b. For RX channels (from MediaLB) MediaLB is the outbound DMA
 - c. Set the channel direction: $RNW = 1$
 - d. Set the channel type: $CT[2:0] = 010$ (asynchronous), 001 (control), 011 (isochronous), or 000 (synchronous)
 - e. Set the channel label: $CL[5:0] = N$
 - f. If $CT[2:0] = 000$ (synchronous), set the mute bit ($MT = 1$)
 - g. Set the channel enable: $CE = 1$
 - h. Set all other bits of the CAT to 0
6. Repeat steps 2 through 5 to initialize all logical channels.

Program the ADT

1. Initialize all bits of the ADT to 0
2. Select a logical channel: $N = 1 - 63$
3. Program the AMBA AHB block ping page for channel N
 - a. Set the 32-bit base address (BA1)
 - b. Set the 11-bit buffer depth (BD1): $BD1 = \text{buffer depth in bytes} - 1$
 - For synchronous channels: $(BD1 + 1) = n \text{ frames per sub-buffer (m) bytes-per-frame (bpf)}$
 - For isochronous channels: $(BD1 + 1) \bmod (BS + 1) = 0$
 - For asynchronous channels: $5 (BD1 + 1) \leq 4096$ (max packet length)
 - For control channels: $5 (BD1 + 1) \leq 4096$ (max packet length)
 - c. For asynchronous and control Tx channels set the packet start bit (PS1) iff the page contains the start of the packet
 - d. Clear the page done bit (DNE1)
 - e. Clear the error bit (ERR1)
 - f. Set the page ready bit (RDY1)
4. Program the AMBA AHB block pong page for channel N
 - a. Set the 32-bit base address (BA2)
 - b. Set the 11-bit buffer depth (BD2): $BD2 = \text{buffer depth in bytes} - 1$
 - For synchronous channels: $(BD2 + 1) = n \text{ frames per sub-buffer (m) bytes-per-frame (bpf)}$
 - For isochronous channels: $(BD2 + 1) \bmod (BS + 1) = 0$
 - For asynchronous channels: $5 (BD2 + 1) \leq 4096$ (max packet length)
 - For control channels: $5 (BD2 + 1) \leq 4096$ (max packet length)
 - c. For asynchronous and control TX channels set the packet start bit (PS2) iff the page contains the start of the packet
 - d. Clear the page done bit (DNE2)
 - e. Clear the error bit (ERR2)
 - f. Set the page ready bit (RDY2)
5. Select Big Endian ($LE = 0$) or Little Endian ($LE = 1$)
6. Select the active page: $PG = 0$ (ping), $PG = 1$ (pong)

7. Set the channel enable (CE) bit for all active logical channels
8. Repeat steps 2 through 7 for all active logical channels.

Service

After initialization, each channel will require periodic servicing. Use the procedures in the following sections to service DMA and MLB interrupts and to poll for MLB system commands.

Servicing the DMA Channel Interrupts

1. Program the [MLB_ACMR0/MLB_ACMR1](#) registers to enable interrupts from all active DMA channels.
2. Select the status clear method. `MLB_ACTL.SCE = 0` (hardware clears on read), `MLB_ACTL.SCE = 1` (software writes a 1 to clear).
3. Select 1 or 2 interrupt signals. Configure the `MLB_ACTL.SMX` bit = 0 (one interrupt for channels 0 through 31 and another interrupt for channels 32 through 63 on). Configure the `MLB_ACTL.SMX` bit = 1 (single interrupt for all channels).
4. Wait for an interrupt.
5. Read the [MLB_ACSR0/MLB_ACSR1](#) registers to determine which channel or channels are causing the interrupt.
6. If the `MLB_ACTL.SCE` bit = 1, write the results of step 5 back to the [MLB_ACSR0](#) and [MLB_ACSR1](#) registers to clear the interrupt.
7. Select a logical channel ($N = 0 - 63$) with an interrupt to service.
8. Read the ADT entry for channel N to:
 - a. Determine the active page (ping or pong) via the PG bit
 - b. Determine which page(s) are done via the DNEN bits
 - c. Determine which channels encountered an AHB error via the ERRn bit
 - d. Determine which asynchronous and control Rx channel pages contain a packet start via the PSn bit (extract the PML)
9. Repeat steps 6 through 8 for all channels with pending interrupts
10. Repeat steps 4 through 9 while there are active channels.

Servicing the MediaLB Status Interrupts

1. Select the MediaLB channel status register ([MLB_MS0/MLB_MS1](#)) to be cleared by software, writing a 0 to the appropriate bits.

2. Program the `MLB_MIEN` register to enable protocol error interrupts for all active MediaLB channels. (`MLB_MIEN.CTXPE = 1`, `MLB_MIEN.CRXPE = 1`, `MLB_MIEN.ATXPE = 1`, `MLB_MIEN.ARXPE = 1`, `MLB_MIEN.SYNCPE = 1`, and `MLB_MIEN.ISOCPE = 1`).
3. Wait for an interrupt on the `MLB_INT0/MLB_INT1` signal.
4. Read the `MLB_MS0/MLB_MS1` registers to determine which channel(s) are causing the interrupt.
5. Read the `RSTS/WSTS` of the appropriate `CDT(s)` to determine the interrupt type.
6. Clear the `RSTS/WSTS` errors to ensure that the current status of channel operations is reflected in the register:

Option	Description
For synchronous RX channels	<code>WSTS[3] = 0</code>
For synchronous TX channels	<code>RSTS[3] = 0</code>
For isochronous RX channels	<code>WSTS[2:1] = 00</code>
For isochronous TX channels	<code>RSTS[2:1] = 00</code>
For asynchronous and control RX channels	<code>WSTS[4] = 0</code> and <code>WSTS[2] = 0</code>
For asynchronous and control TX channels	<code>RSTS[4] = 0</code> and <code>RSTS[2] = 0</code>

Polling for MediaLB System Commands

The MediaLB System status (`MLB_MSS`) register is used to detect a system command received from the MediaLB controller. The processor's peripheral automatically sends the appropriate system response to the MediaLB controller. The procedure for the application is:

1. The application periodically polls the `MLB_MSS` register.
2. Clear by writing a 0 to the appropriate bit in the `MLB_MSS` register after the application finishes the service.
3. If `MLB_MSS.SWSYSCMD = 1`, read the `MLB_MSD` register to receive the system data sent from MediaLB controller.

ADSP-2156x MLB Register Descriptions

MediaLB Device Interface Macro 2 (MLB) contains the following registers.

Table 20-19: ADSP-2156x MLB Register List

Name	Description
<code>MLB_ACMR0</code>	Peripheral Channel Mask 0 Register
<code>MLB_ACMR1</code>	Peripheral Channel Mask 1 Register
<code>MLB_ACSR0</code>	Peripheral Channel Status 0 Register
<code>MLB_ACSR1</code>	Peripheral Channel Status 1 Register

Table 20-19: ADSP-2156x MLB Register List (Continued)

Name	Description
MLB_ACTL	Bus Control Register
MLB_CTL0	MediaLB Control 0 Register
MLB_CTL1	Control 1 Register
MLB_GCTL	MLB Global Control Register
MLB_HCBR0	HBI Channel Busy 0 Register
MLB_HCBR1	HBI Channel Busy 1 Register
MLB_HCER0	HBI Channel Error 0 Register
MLB_HCER1	HBI Channel Error 1 Register
MLB_HCMR0	HBI Channel Mask 0 Register
MLB_HCMR1	HBI Channel Mask 1 Register
MLB_HCTL	HBI Control Register
MLB_MADR	Memory Interface Address Register
MLB_MCTL	Memory Interface Control Register
MLB_MDAT0	Memory Interface Control Data 0 Register
MLB_MDAT1	Memory Interface Control Data 1 Register
MLB_MDAT2	Memory Interface Control Data 2 Register
MLB_MDAT3	Memory Interface Control Data 3 Register
MLB_MDWE0	Memory Interface Control Data Write Enable 0 Register
MLB_MDWE1	Memory Interface Control Data Write Enable 1 Register
MLB_MDWE2	Memory Interface Control Data Write Enable 2 Register
MLB_MDWE3	Memory Interface Control Data Write Enable 3 Register
MLB_MIEN	Interrupt Enable Register
MLB_MS0	Channel Status 0 Register
MLB_MS1	Channel Status 1 Register
MLB_MSD	System Data Register
MLB_MSS	System Status Register

Peripheral Channel Mask 0 Register

The `MLB_ACMR0` register allows control over which channel(s) generate interrupts on `MLB_INT[1:0]`. All of the bits in this register default to 0 (masked) so the HC must initially write to the `MLB_ACMR0` register to enable interrupts. Each bit of this register is read/write accessible.

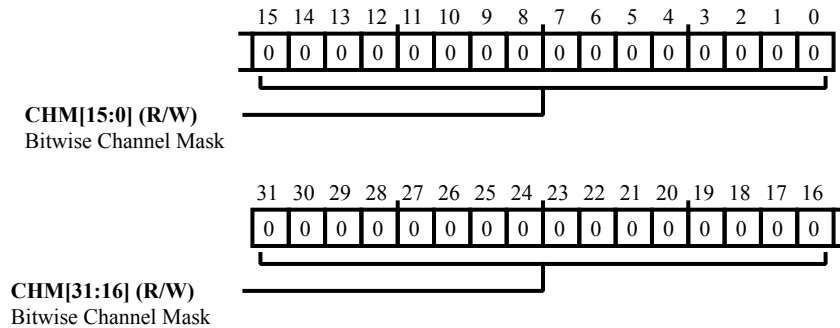


Figure 20-7: `MLB_ACMR0` Register Diagram

Table 20-20: `MLB_ACMR0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Bitwise Channel Mask. The <code>MLB_ACMR0.CHM</code> bit field masks or unmasks channels 31-0.
		0 Mask interrupt for Channel 31:0 (bitwise; all channels shown)
		4294967295 Unmask interrupt for Channel 31:0 (bitwise; all channels shown)

Peripheral Channel Mask 1 Register

The `MLB_ACMR1` register allows control over which channel(s) generate interrupts on `MLB_INT[1:0]`. All of the bits in this register default to 0 (masked) so the HC must initially write to the `MLB_ACMR1` register to enable interrupts. Each bit of this register is read/write accessible.

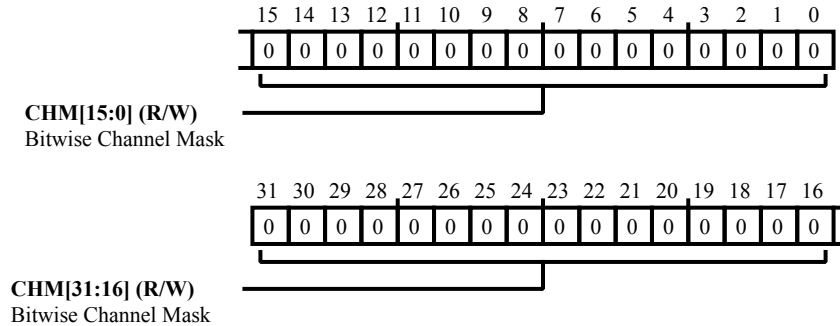


Figure 20-8: `MLB_ACMR1` Register Diagram

Table 20-21: `MLB_ACMR1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Bitwise Channel Mask. The <code>MLB_ACMR1.CHM</code> bit field masks or unmask channels 63:32.
		0 Mask interrupt for Channel 63:32 (bitwise; all channels shown)
		4294967295 Unmask interrupt for Channel 63:32 (bitwise; all channels shown)

Peripheral Channel Status 0 Register

The `MLB_ACSR0` register contains interrupt bits for each of the 64 physical channels. When a bit in this register is set, it indicates that the corresponding physical channel has an interrupt pending.

A peripheral interrupt is triggered when either `DNEn` or `ERRn` is set within the Bus Channel Descriptor. The HC is notified of the channel interrupt via `MLB_INT[1:0]`. When an interrupt occurs in `ACCUSER` (for channels 31 to 0) `MLB_INT[0]` is set. When an interrupt occurs in `MLB_ACSR1` (for channels 63 to 32) `MLB_INT[1]` is set.

Interrupts in the `MLB_ACSR0` and `MLB_ACSR1` registers can be optionally multiplexed onto a single interrupt signal, `MLB_INT[0]`, if the `MLB_ACTL.SMX` bit = 1.

If the `MLB_ACTL.SCE` bit = 0, hardware automatically clears the interrupt bit(s) after the HC reads the peripheral channel status registers. Alternatively, if the `MLB_ACTL.SCE` bit = 1, software must write a 1 to the appropriate bit(s) of the peripheral channel status registers to clear the interrupt(s).

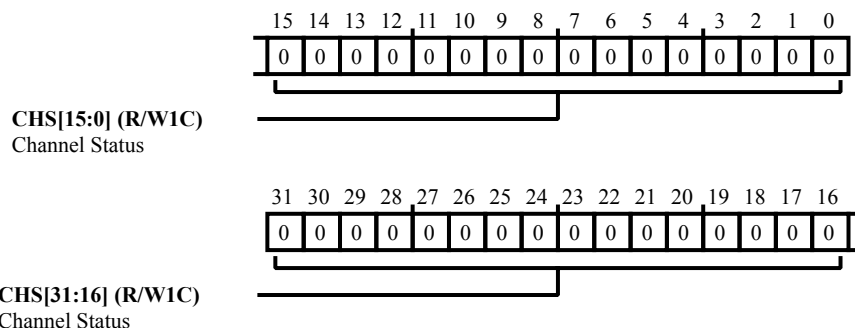


Figure 20-9: `MLB_ACSR0` Register Diagram

Table 20-22: `MLB_ACSR0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1C)	CHS	Channel Status. The <code>MLB_ACSR0.CHS</code> bit field indicates channel status for channels 31-0.
		0 No interrupt on Channel 31:0 (bitwise; all channels shown)
		4294967295 Interrupt on Channel 31:0 (bitwise; all channels shown)

Peripheral Channel Status 1 Register

The `MLB_ACSR1` register contains interrupt bits for each of the 64 physical channels. When a bit in this register is set, it indicates that the corresponding physical channel has an interrupt pending.

A peripheral interrupt is triggered when either `DNEn` or `ERRn` is set within the Bus Channel Descriptor. The HC is notified of the channel interrupt via `MLB_INT[1:0]`. When an interrupt occurs in `ACCUSER` (for channels 31 to 0) `MLB_INT[0]` is set. When an interrupt occurs in `MLB_ACSR1` (for channels 63 to 32) `MLB_INT[1]` is set.

Interrupts in the `MLB_ACSR1` and `MLB_ACSR0` registers can be optionally multiplexed onto a single interrupt signal, `MLB_INT[0]`, if the `MLB_ACTL.SMX` bit = 1.

If the `MLB_ACTL.SCE` bit = 0, hardware automatically clears the interrupt bit(s) after the HC reads the peripheral channel status registers. Alternatively, if the `MLB_ACTL.SCE` bit = 1, software must write a 1 to the appropriate bit(s) of peripheral channel status registers to clear the interrupt(s).

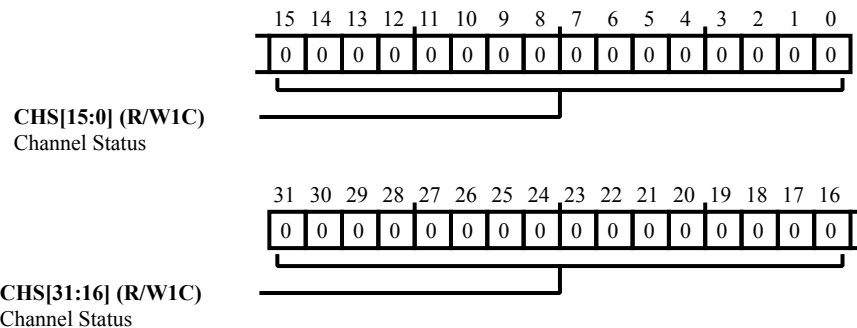


Figure 20-10: `MLB_ACSR1` Register Diagram

Table 20-23: `MLB_ACSR1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1C)	CHS	Channel Status. The <code>MLB_ACSR1.CHS</code> bit field indicates channel status for channels 63-32.
		0 No interrupt on Channel 63:32 (bitwise; all channels shown)
		4294967295 Interrupt on Channel 63:32 (bitwise; all channels shown)

Bus Control Register

The `MLB_ACTL` register is written by the HC to configure the block for channel interrupts. This register contains bits to select the packet buffering mode and the DMA mode. This register also contains bits that are used to multiplex channel interrupts onto a single interrupt signal and to select the method of clearing channel interrupts.

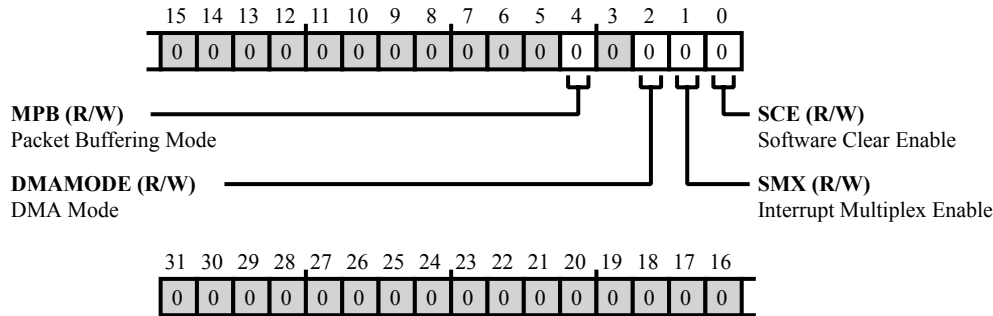


Figure 20-11: `MLB_ACTL` Register Diagram

Table 20-24: `MLB_ACTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	MPB	Packet Buffering Mode. The <code>MLB_ACTL.MPB</code> bit selects whether the buffering mode is single-packet or multiple-packet.
		0 single-packet mode
		1 multiple-packet mode
2 (R/W)	DMAMODE	DMA Mode. The <code>MLB_ACTL.DMAMODE</code> bit selects between DMA mode 1 and 0. DMA Mode 0 uses incrementing bursts of an unspecified length. This allows the block to perform single beat transfers as well as an incrementing (INCR) burst of unspecified length up to the maximum specified burst length (8 beats). DMA Mode 1 uses only INCR8, INCR4, and SINGLE beat bursts. The <code>hburst[2:0]</code> signal encodes the burst transfer used for a DMA operation.
		0 DMA Mode 0
		1 DMA Mode 1

Table 20-24: MLB_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	SMX	Interrupt Multiplex Enable. The <code>MLB_ACTL.SMX</code> bit selects whether <code>ACSR0</code> generates an interrupt on <code>MLB_INT[0]</code> and <code>ACSR1</code> generates an interrupt on <code>MLB_INT[1]</code> or <code>ACSR0</code> and <code>ACSR1</code> generate an interrupt on <code>MLB_INT[0]</code> only.
		0 <code>ACSR0</code> generates an interrupt on <code>MLB_INT[0]</code> ; <code>ACSR1</code> generates an interrupt on <code>MLB_INT[1]</code>
		1 <code>ACSR0</code> and <code>ACSR1</code> generate an interrupts on <code>MLB_INT[0]</code> only
0 (R/W)	SCE	Software Clear Enable. The <code>MLB_ACTL.SCE</code> bit selects whether hardware clears the interrupt after a <code>ACSRn</code> register read or software clears the interrupt.
		0 Hardware clears interrupt after a <code>ACSRn</code> register read
		1 Software clears interrupt

MediaLB Control 0 Register

The `MLB_CTL0` register contains bit that enable the module and provide module status. Note that the maximum speed for ML3-pin mode is 1024 x FS

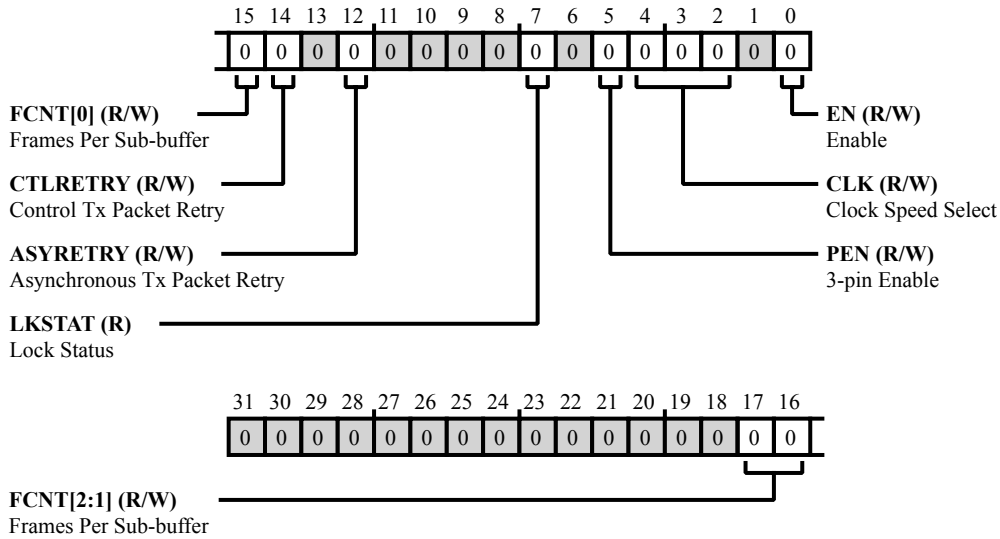


Figure 20-12: `MLB_CTL0` Register Diagram

Table 20-25: `MLB_CTL0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	FCNT	Frames Per Sub-buffer. The <code>MLB_CTL0.FCNT</code> bit field configures the frames per sub-buffer on synchronous channels.	
		0	1 Frame per sub-buffer (Operation is the same as Standard mode)
		1	2 frames per sub-buffer
		2	4 frames per sub-buffer
		3	8 frames per sub-buffer
		4	16 frames per sub-buffer
		5	32 frames per sub-buffer
		6	64 frames per sub-buffer
14 (R/W)	CTLRETRY	Control Tx Packet Retry. When the <code>MLB_CTL0.CTLRETRY</code> bit is set, a control packet that is flagged with a Break or Protocol error by the receiver is retransmitted. When cleared, a control packet that is flagged with a Break or Protocol error by the receiver is skipped.	

Table 20-25: MLB_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	ASYRETRY	Asynchronous Tx Packet Retry. When the MLB_CTL0 . ASYRETRY bit is set, an asynchronous packet that is flagged with a Break or Protocol error by the receiver is retransmitted. When cleared, an asynchronous packet that is flagged with a Break or ProtocolError by the receiver is skipped.
7 (R/NW)	LKSTAT	Lock Status. When the MLB_CTL0 . LKSTAT bit is set (=0), the MediaLB block is synchronized to the incoming MediaLB frame with the following conditions. <ul style="list-style-type: none"> • If MLB_CTL1 . LOCK =0 (unlocked), MLB_CTL1 . LOCK is set after a FRAME-SYNC is detected at the same position for three consecutive frames. • If MLB_CTL1 . LOCK =1 (locked), MLB_CTL1 . LOCK is cleared after not receiving a FRAMESYNC at the expected time for two consecutive frames. In this case FRAMESYNC patterns occurring at locations other than the expected one are ignored.
5 (R/W)	PEN	3-pin Enable. The MLB_CTL0 . PEN bit configures the 3-pin mode.
		0 3-pin interface enabled 1 invalid condition
4:2 (R/W)	CLK	Clock Speed Select. The MLB_CTL0 . CLK bit field sets the clock speed.
		0 256 x Fs (MLB_CTL.PEN = 0)
		1 512 x Fs (MLB_CTL.PEN = 0)
2 1024 x Fs (MLB_CTL.PEN = 0)		
0 (R/W)	EN	Enable. When the MLB_CTL0 . EN bit is set (=1), MediaLB clock, signal, and data are received and transmitted on the appropriate MediaLB pins.

Control 1 Register

The `MLB_CTL1` register contains bits that provide lock status and control system commands.

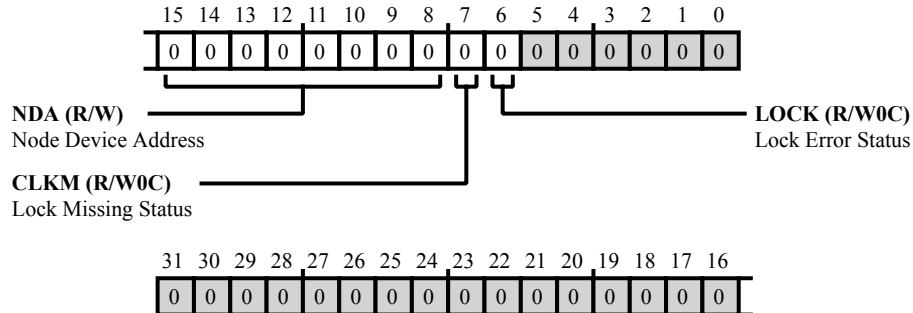


Figure 20-13: `MLB_CTL1` Register Diagram

Table 20-26: `MLB_CTL1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	NDA	Node Device Address. The <code>MLB_CTL1 . NDA</code> bit field is used for system commands directed to individual MediaLB nodes.
7 (R/W0C)	CLKM	Lock Missing Status. The <code>MLB_CTL1 . CLKM</code> bit is set when the MediaLB clock is not toggling at the pin. This bit is cleared by software.
6 (R/W0C)	LOCK	Lock Error Status. The <code>MLB_CTL1 . LOCK</code> bit is set when the MediaLB is unlocked. This bit is cleared by software.

MLB Global Control Register

The `MLB_GCTL` register contains bits that manage the MLB clock.

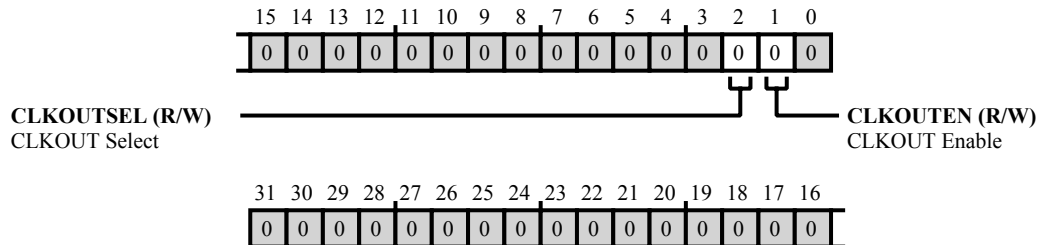


Figure 20-14: `MLB_GCTL` Register Diagram

Table 20-27: `MLB_GCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	CLKOUTSEL	CLKOUT Select. The <code>MLB_GCTL.CLKOUTSEL</code> bit selects MLB 3 pin clock
		0 MLB 3 pin clock is selected for <code>MLBCLKOUT</code>
1 (R/W)	CLKOUTEN	CLKOUT Enable.

HBI Channel Busy 0 Register

The HC can determine which channel(s) are busy by reading the `MLB_HCBR0` register. An HBI channel is busy if:

- it is currently loaded into one of the two AGUs
- the channel is enabled, `CE = 1` from the Channel Allocation Table
- the DMA is active

When an HBI channel is busy, hardware may write back its local copy of the channel descriptor at any time. System software should not write a CDT descriptor for a channel that is busy. Only two HBI channels can be busy at any given time. Each bit of this register is read-only.

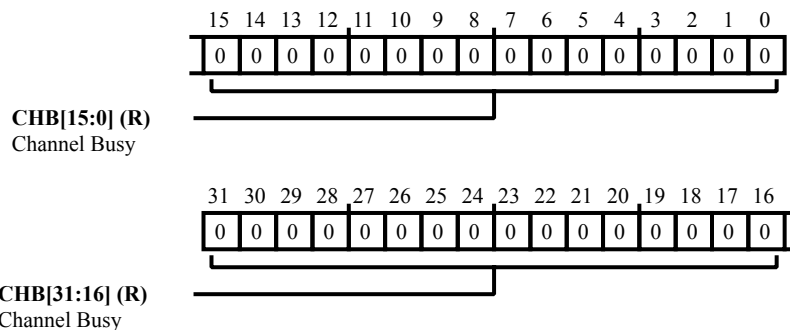


Figure 20-15: `MLB_HCBR0` Register Diagram

Table 20-28: `MLB_HCBR0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CHB	Channel Busy. The <code>MLB_HCBR0.CHB</code> bit field contains the bitwise channel busy bit for channels 31:0. When a bit is cleared (=0) the channel is idle. When a bit is set (=1) the channel is busy.

HBI Channel Busy 1 Register

The HC can determine which channel(s) are busy by reading the `MLB_HCBR1` register. An HBI channel is busy if:

- it is currently loaded into one of the two AGUs
- the channel is enabled, `CE = 1` from the Channel Allocation Table
- the DMA is active

When an HBI channel is busy, hardware may write back its local copy of the channel descriptor at any time. System software should not write a CDT descriptor for a channel that is busy. Only two HBI channels can be busy at any given time. Each bit of this register is read-only.

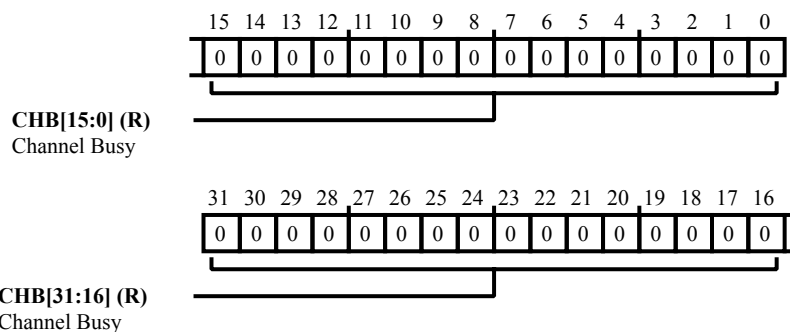


Figure 20-16: `MLB_HCBR1` Register Diagram

Table 20-29: `MLB_HCBR1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CHB	Channel Busy. The <code>MLB_HCBR1 . CHB</code> bit field contains the bitwise channel busy bit for channels 63:32. When a bit is cleared (=0) the channel is idle. When a bit is set (=1) the channel is busy.

HBI Channel Error 0 Register

The `MLB_HCER0` register indicates which channels (channels 31:0) have encountered fatal errors.

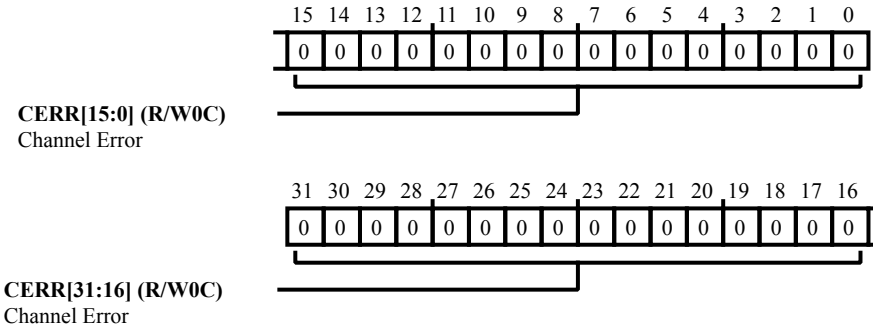


Figure 20-17: `MLB_HCER0` Register Diagram

Table 20-30: `MLB_HCER0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	CERR	Channel Error. The <code>MLB_HCER0.CERR</code> bit field reports bitwise errors for channels 31:0.

HBI Channel Error 1 Register

The `MLB_HCER1` register indicates which channel(s) have encountered fatal errors.

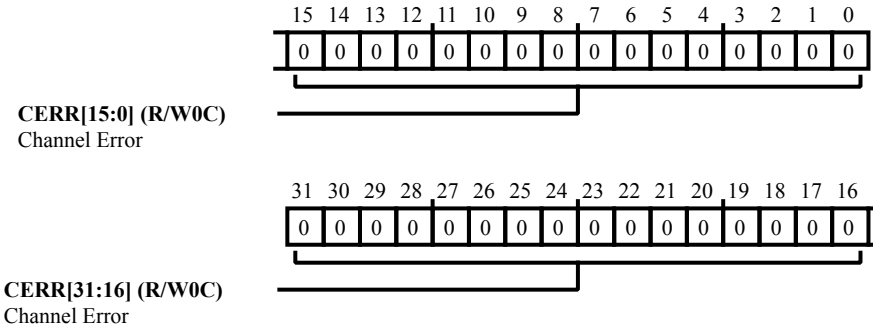


Figure 20-18: `MLB_HCER1` Register Diagram

Table 20-31: `MLB_HCER1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	CERR	Channel Error. The <code>MLB_HCER1 . CERR</code> bite field reports bitwise errors for channels 63:32.

HBI Channel Mask 0 Register

The `MLB_HCMR0` register controls which channels (for channels 31:0) are able to generate an HBI interrupt.

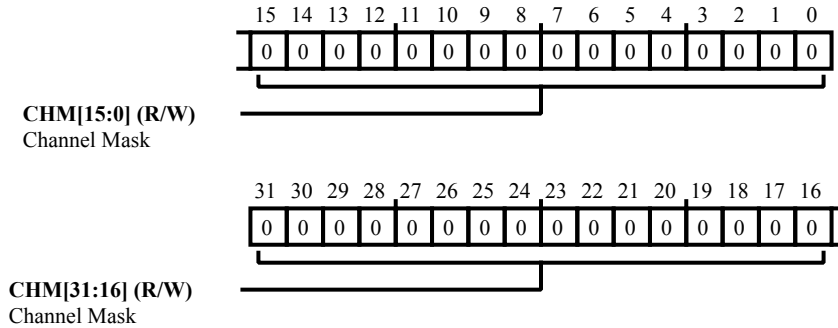


Figure 20-19: `MLB_HCMR0` Register Diagram

Table 20-32: `MLB_HCMR0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Channel Mask. The <code>MLB_HCMR0.CHM</code> bit field contains the bitwise channel mask bit for channels 31:0. When a bit is cleared (=0) the channel is masked. When set (=1) the channel is unmasked.

HBI Channel Mask 1 Register

The `MLB_HCMR1` register controls which channels (for channels 63:32) are able to generate an HBI interrupt.

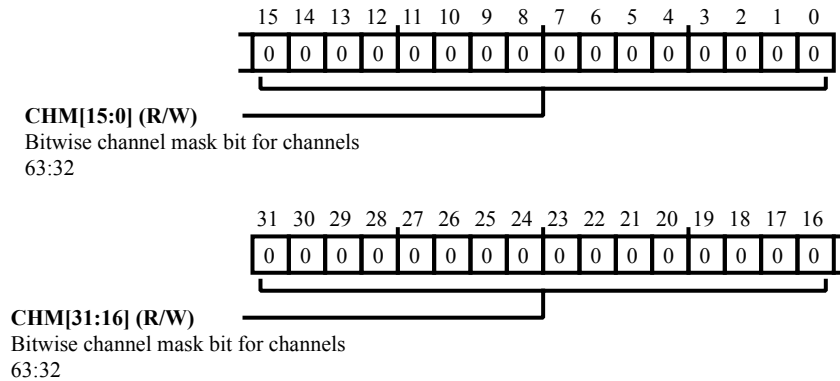


Figure 20-20: `MLB_HCMR1` Register Diagram

Table 20-33: `MLB_HCMR1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Bitwise channel mask bit for channels 63:32. The <code>MLB_HCMR1.CHM</code> bit field contains the bitwise channel mask bit for channels 63:32. When a bit is cleared (=0) the channel is masked. When set (=1) the channel is unmasked.

HBI Control Register

The `MLB_HCTL` register controls and monitors general operation of the HBI block through the Address Generation Units) by reading and writing the register through the I/O interface. Each bit of this register is read/write.

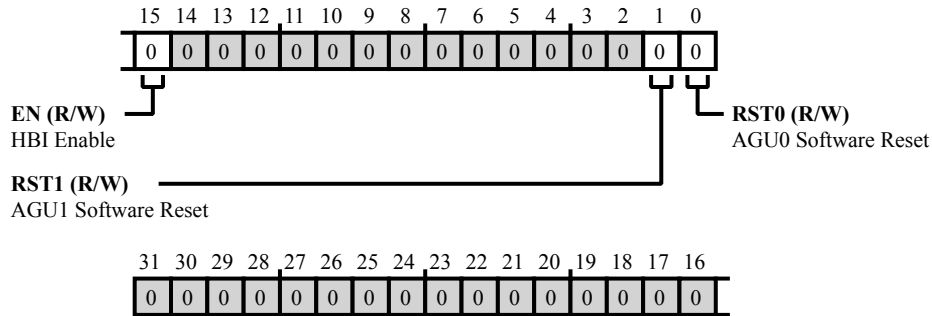


Figure 20-21: `MLB_HCTL` Register Diagram

Table 20-34: `MLB_HCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	HBI Enable. Setting the <code>MLB_HCTL.EN</code> bit enables the HBI.
1 (R/W)	RST1	AGU1 Software Reset. Setting the <code>MLB_HCTL.RST1</code> bit resets AGU1.
0 (R/W)	RST0	AGU0 Software Reset. Setting the <code>MLB_HCTL.RST0</code> bit resets AGU0.

Memory Interface Address Register

The `MLB_MADR` register contains bit fields that contain the Channel Table RAM (CTR) or Data Buffer RAM (DBR) addresses. It also contains bits that set target location and read-not-write parameters.

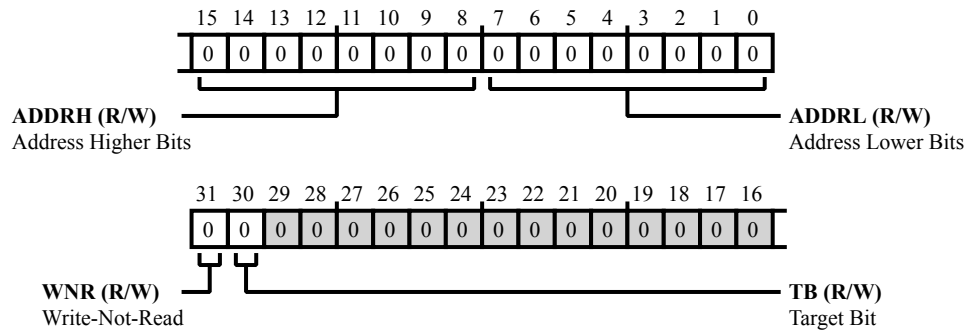


Figure 20-22: `MLB_MADR` Register Diagram

Table 20-35: `MLB_MADR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	WNR	Write-Not-Read. The <code>MLB_MADR.WNR</code> bit selects Write-Not-Read.
		0 Read
		1 Write
30 (R/W)	TB	Target Bit. The <code>MLB_MADR.TB</code> bit sets the target location.
		0 Channel Table RAM (CTR)
		1 Data Buffer RAM (DBR)
15:8 (R/W)	ADDRH	Address Higher Bits. The <code>MLB_MADR.ADDRH</code> bit field contains the DBR address of the 8-bit entry (bits [13:8]).
7:0 (R/W)	ADDRL	Address Lower Bits. The <code>MLB_MADR.ADDRL</code> bit field contains the CTR address of the 128-bit entry or the DBR address of the 8-bit entry (bits [7:0]).

Memory Interface Control Register

The `MLB_MCTL` register contains a bit that indicates that the data transfer is complete.

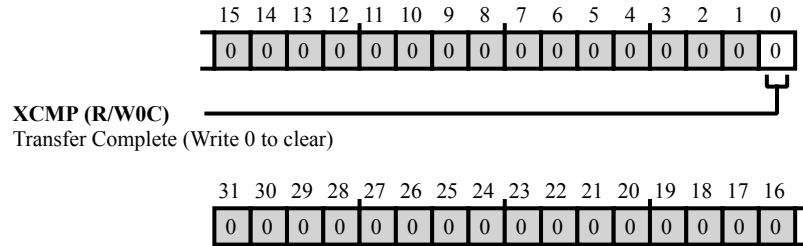


Figure 20-23: `MLB_MCTL` Register Diagram

Table 20-36: `MLB_MCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W0C)	XCMP	Transfer Complete (Write 0 to clear). The <code>MLB_MCTL.XCMP</code> bit indicates that the data transfer is complete.

Memory Interface Control Data 0 Register

The `MLB_MDAT0` register contains Channel Table RAM (CTR) data (bits [31:0] of 128-bit entry) or Data Buffer RAM (DBR) data (bits [7:0] of 8-bit entry).

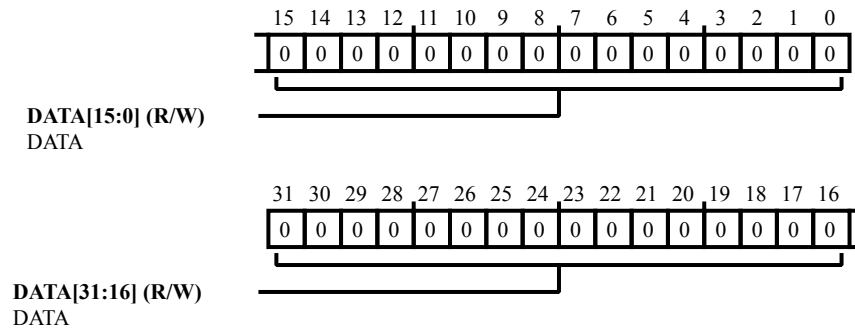


Figure 20-24: `MLB_MDAT0` Register Diagram

Table 20-37: `MLB_MDAT0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The <code>MLB_MDAT0 . DATA</code> bit field contains CTR data or DBR data.

Memory Interface Control Data 1 Register

The `MLB_MDAT1` register contains Channel Table RAM (CTR) data (bits [63:32] of 128-bit entry).

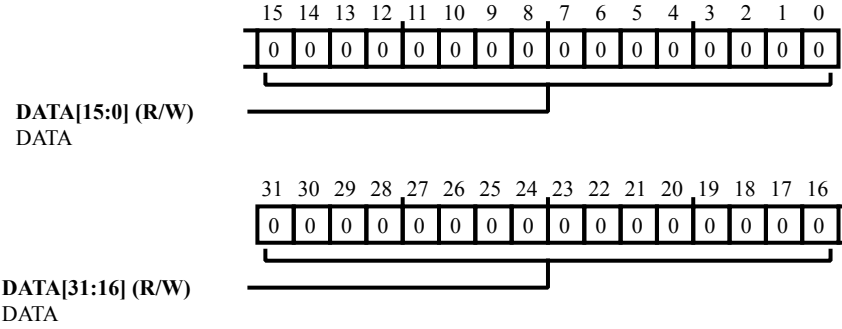


Figure 20-25: `MLB_MDAT1` Register Diagram

Table 20-38: `MLB_MDAT1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The <code>MLB_MDAT1 . DATA</code> bit field contains CTR data.

Memory Interface Control Data 2 Register

The `MLB_MDAT2` register contains Channel Table RAM (CTR) data (bits [95:64] of 128-bit entry).

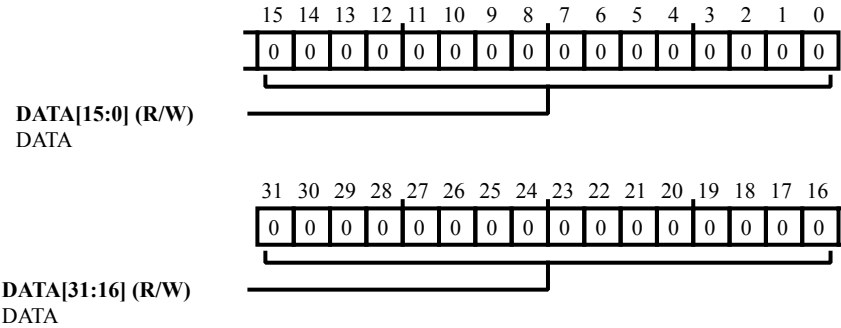


Figure 20-26: `MLB_MDAT2` Register Diagram

Table 20-39: `MLB_MDAT2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The <code>MLB_MDAT2</code> .DATA bit field contains CTR data.

Memory Interface Control Data 3 Register

The `MLB_MDAT3` register contains Channel Table RAM (CTR) data (bits [127:96] of 128-bit entry).

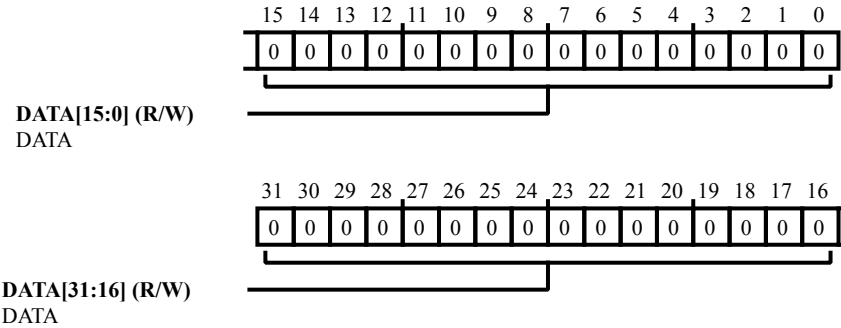


Figure 20-27: `MLB_MDAT3` Register Diagram

Table 20-40: `MLB_MDAT3` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The <code>MLB_MDAT3 . DATA</code> bit field contains CTR data.

Memory Interface Control Data Write Enable 0 Register

The `MLB_MDWE0` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [31:0]. When cleared (=0) the bit is disabled, when set (=1) the bit is enabled.

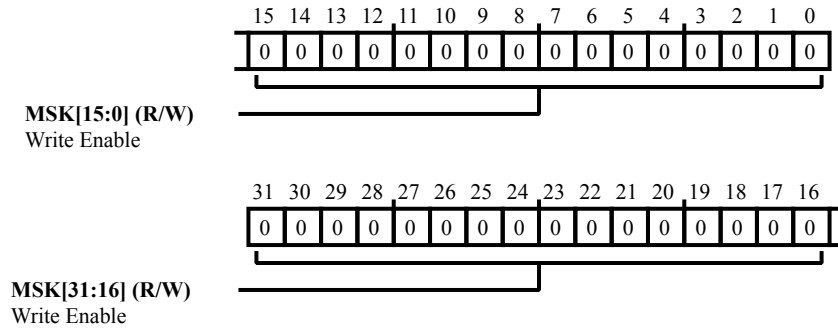


Figure 20-28: MLB_MDWE0 Register Diagram

Table 20-41: MLB_MDWE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Write Enable. The <code>MLB_MDWE0</code> .MSK bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 1 Register

The `MLB_MDWE1` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [63:32]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

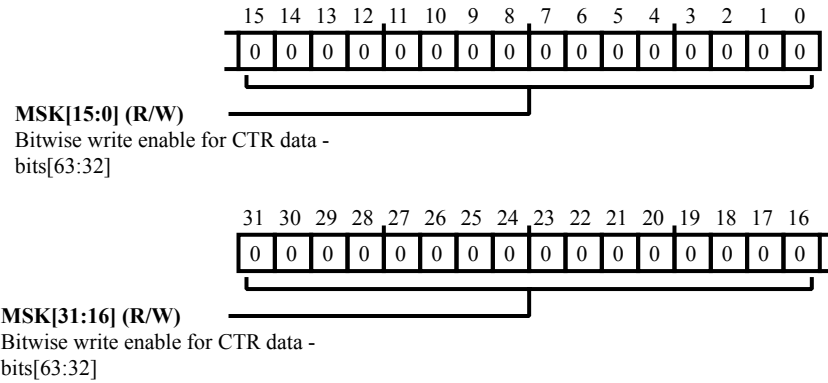


Figure 20-29: MLB_MDWE1 Register Diagram

Table 20-42: MLB_MDWE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[63:32]. The <code>MLB_MDWE1</code> . MSK bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 2 Register

The `MLB_MDWE2` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [95:64]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

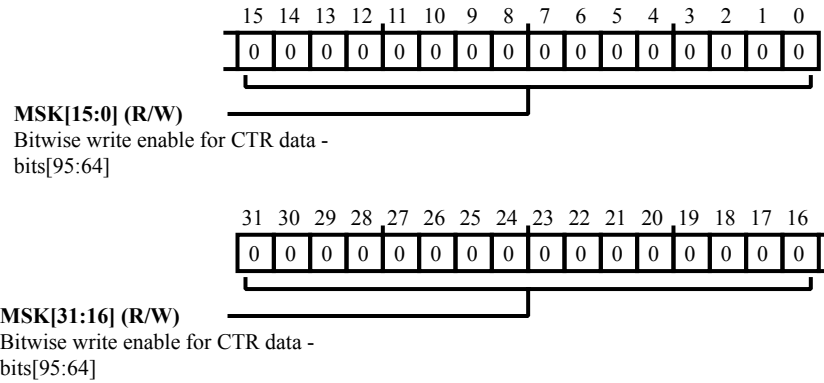


Figure 20-30: MLB_MDWE2 Register Diagram

Table 20-43: MLB_MDWE2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[95:64]. The <code>MLB_MDWE2</code> .MSK bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 3 Register

The `MLB_MDWE3` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [127:96]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

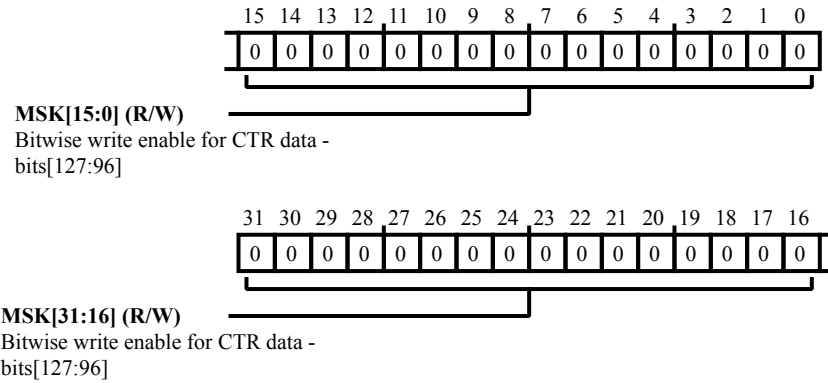


Figure 20-31: `MLB_MDWE3` Register Diagram

Table 20-44: `MLB_MDWE3` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[127:96]. The <code>MLB_MDWE3</code> .MSK bit field contains the bitwise write enable for CTR data.

Interrupt Enable Register

The `MLB_MIEN` register is used to enable various interrupt conditions.

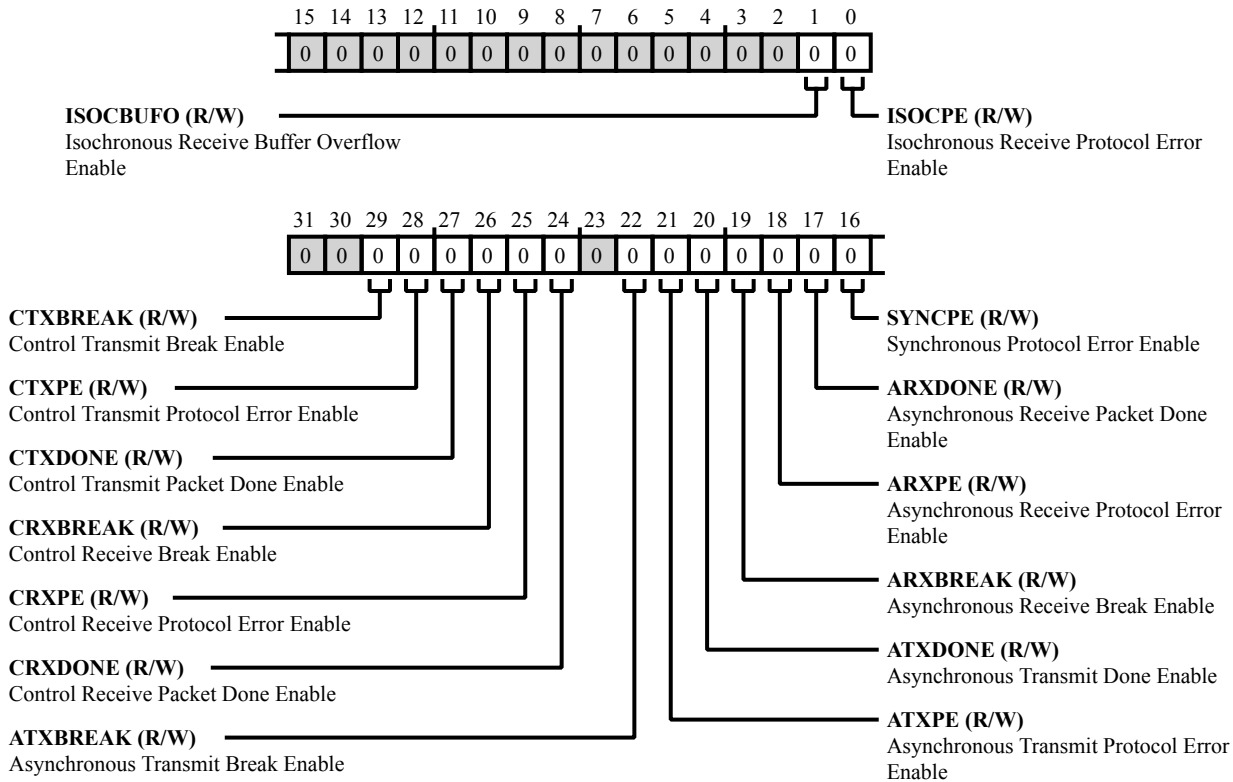


Figure 20-32: `MLB_MIEN` Register Diagram

Table 20-45: `MLB_MIEN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	CTXBREAK	Control Transmit Break Enable. When the <code>MLB_MIEN.CTXBREAK</code> bit is set, a ReceiverBreak response received from the receiver on a control Tx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set.
28 (R/W)	CTXPE	Control Transmit Protocol Error Enable. When the <code>MLB_MIEN.CTXPE</code> bit is set, a ProtocolError generated by the receiver on a control Tx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set.
27 (R/W)	CTXDONE	Control Transmit Packet Done Enable. When the <code>MLB_MIEN.CTXDONE</code> bit is set, a packet transmitted with no errors on a control Tx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set.

Table 20-45: MLB_MIEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	CRXBREAK	Control Receive Break Enable. When the MLB_MIEN.CRXBREAK bit is set, a ControlBreak command received from the transmitter on a control Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
25 (R/W)	CRXPE	Control Receive Protocol Error Enable. When the MLB_MIEN.CRXPE bit is set, a ProtocolError detected on a control Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
24 (R/W)	CRXDONE	Control Receive Packet Done Enable. When the MLB_MIEN.CRXDONE bit is set, a packet received with no errors on a control Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
22 (R/W)	ATXBREAK	Asynchronous Transmit Break Enable. When the MLB_MIEN.ATXBREAK bit is set, a ReceiverBreak response received from the receiver on an asynchronous Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
21 (R/W)	ATXPE	Asynchronous Transmit Protocol Error Enable. When the MLB_MIEN.ATXPE bit is set, a ProtocolError generated by the receiver on an asynchronous Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
20 (R/W)	ATXDONE	Asynchronous Transmit Done Enable. When the MLB_MIEN.ATXDONE bit is set, a packet transmitted with no errors on an asynchronous Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
19 (R/W)	ARXBREAK	Asynchronous Receive Break Enable. When the MLB_MIEN.ARXBREAK bit is set, a AsyncBreak command received from the transmitter on an asynchronous Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
18 (R/W)	ARXPE	Asynchronous Receive Protocol Error Enable. When the MLB_MIEN.ARXPE bit is set, a protocol error detected on an asynchronous Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
17 (R/W)	ARXDONE	Asynchronous Receive Packet Done Enable. When the MLB_MIEN.ARXDONE bit is set, a packet received with no errors on an asynchronous Rx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.

Table 20-45: MLB_MIEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SYNCPE	Synchronous Protocol Error Enable. When the <code>MLB_MIEN.SYNCPE</code> bit is set, a protocol error detected on a synchronous Rx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set. This occurs only when isochronous flow control is disabled.
1 (R/W)	ISOCBUFO	Isochronous Receive Buffer Overflow Enable. When the <code>MLB_MIEN.ISOCBUFO</code> bit is set, a buffer overflow on an isochronous Rx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set. This occurs only when isochronous flow control is disabled.
0 (R/W)	ISOCPE	Isochronous Receive Protocol Error Enable. When the <code>MLB_MIEN.ISOCPE</code> bit is set, a protocol error detected on an isochronous Rx channel causes the appropriate channel bit in the <code>MLB_MS0</code> or <code>MLB_MS1</code> registers to be set.

Channel Status 0 Register

The `MLB_MS0` register indicates the channel status for MediaLB channels 31 to 0. Channel status bits are set by hardware and cleared by software. Status is only set if the appropriate bits in the `MLB_MIEN` register are set.

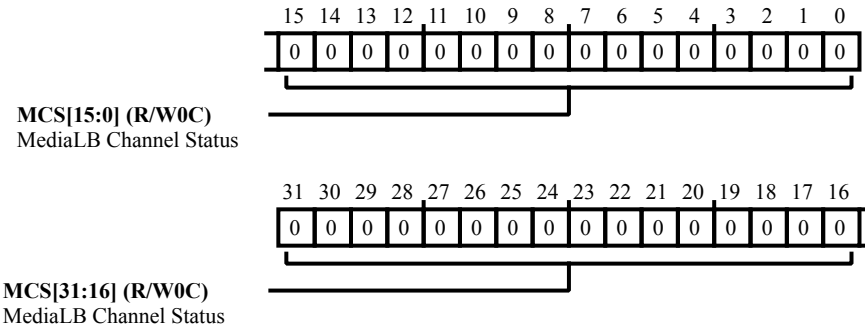


Figure 20-33: `MLB_MS0` Register Diagram

Table 20-46: `MLB_MS0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	MCS	MediaLB Channel Status. The <code>MLB_MS0.MCS</code> bit field indicates the MediaLB channel status for channels 31 to 0.

Channel Status 1 Register

The `MLB_MS1` register indicates the channel status for MediaLB channels 32 to 63. Channel status bits are set by hardware and cleared by software. Status is only set if the appropriate bits in the `MLB_MIEN` register are set.

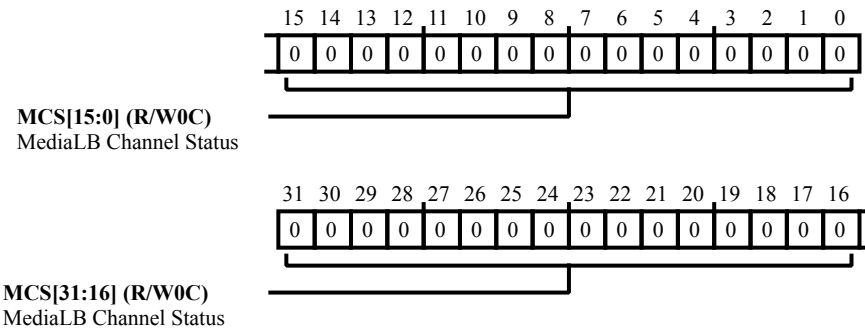


Figure 20-34: `MLB_MS1` Register Diagram

Table 20-47: `MLB_MS1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	MCS	MediaLB Channel Status. The <code>MLB_MS1.MCS</code> bit field indicates the MediaLB channel status for channels 63 to 32.

System Data Register

The `MLB_MSD` register allows system software to receive control information from the MLB controller. The `MLB_MSD` register is updated once per frame by the hardware during the MLB system channel.

The `MLB_MSD` register is loaded with the data from the `MLBDAT_IN` signal during the system channel quadlet. System software must read this register before the start of the next MLB frame to prevent the current data from being lost.

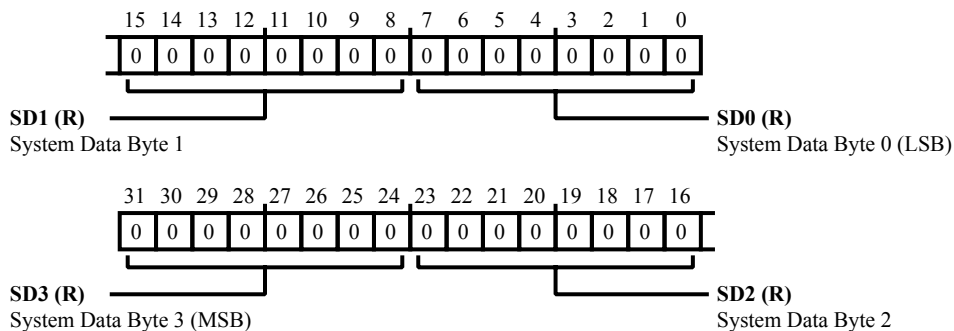


Figure 20-35: `MLB_MSD` Register Diagram

Table 20-48: `MLB_MSD` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	SD3	System Data Byte 3 (MSB). The <code>MLB_MSD.SD3</code> bits are updated with MediaLB Data [31:24] when a MediaLB software system command is received in the system quadlet. If the <code>MLB_MSS.SWSYSCMD</code> bit is already set, then SD3 is not updated.
23:16 (R/NW)	SD2	System Data Byte 2. The <code>MLB_MSD.SD2</code> bits are updated with MediaLB Data [23:16] when a MediaLB software system command is received in the system quadlet. If the <code>MLB_MSS.SWSYSCMD</code> bit is already set, then SD2 is not updated.
15:8 (R/NW)	SD1	System Data Byte 1. The <code>MLB_MSD.SD1</code> bits are updated with MediaLB Data [15:8] when a MediaLB software system command is received in the system quadlet. If the <code>MLB_MSS.SWSYSCMD</code> bit is already set, then SD1 is not updated.
7:0 (R/NW)	SD0	System Data Byte 0 (LSB). The <code>MLB_MSD.SD0</code> bits are updated with MediaLB Data [7:0] when a MediaLB software system command is received in the system quadlet. If the <code>MLB_MSS.SWSYSCMD</code> bit is already set, then SD0 is not updated.

System Status Register

The `MLB_MSS` register allows system software to monitor and control the status of the MLB network. The register is updated once per frame by hardware during the MLB system channel. The bits of this register are not valid until the processor is locked to the MLB interface (except for the bits associated with MLB lock and unlock, SDMU and SDML). System software must service events before the start of the next MLB frame to prevent the current frame status from being lost.

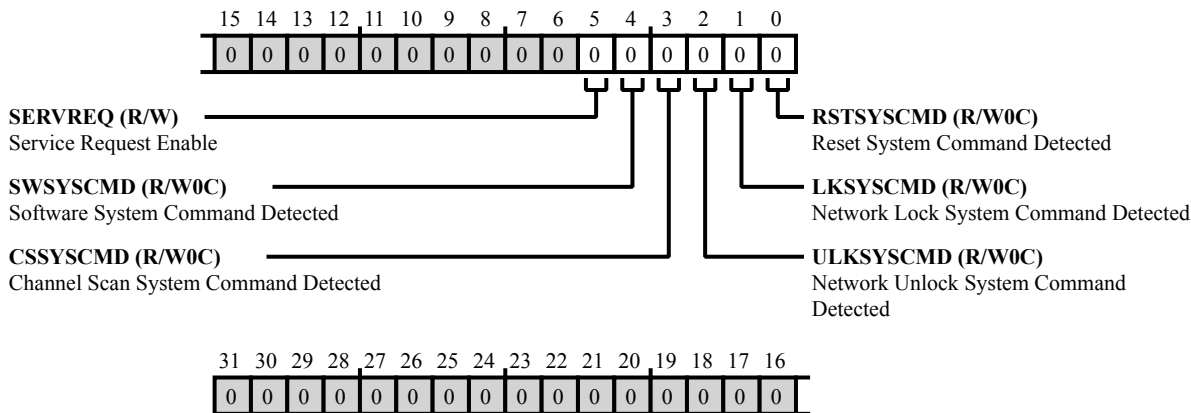


Figure 20-36: MLB_MSS Register Diagram

Table 20-49: MLB_MSS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SERVREQ	Service Request Enable. When the <code>MLB_MSS.SERVREQ</code> bit set, the MediaLB block responds with a device present, request service system response if a matching channel scan system command is detected. When cleared, the MediaLB block responds with a device present system response.
4 (R/W0C)	SWSYSCMD	Software System Command Detected. The <code>MLB_MSS.SWSYSCMD</code> bit indicates that a software system command was detected (in the system quadlet). The <code>MLB_MSS.SWSYSCMD</code> bit is set by hardware, cleared by software. Data is stored in the <code>MLB_MSD</code> register for this command.
3 (R/W0C)	CSSYSCMD	Channel Scan System Command Detected. The <code>MLB_MSS.CSSYSCMD</code> bit indicates that a channel scan system command was detected (in the system quadlet). The <code>MLB_MSS.CSSYSCMD</code> bit is set by hardware, cleared by software. If the node address specified in the data quadlet matches the value in the <code>MLB_CTL1.NDA</code> bit field, the device responds either device present or device present, request service system response in the next system quadlet.

Table 20-49: MLB_MSS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W0C)	ULKSYSCMD	Network Unlock System Command Detected. The MLB_MSS . ULKSYSCMD bit indicates a network unlock system command was detected (in the system quadlet). The MLB_MSS . ULKSYSCMD bit is by hardware, cleared by software.
1 (R/W0C)	LKSYSCMD	Network Lock System Command Detected. The MLB_MSS . LKSYSCMD bit indicates a network lock system command was detected (in the system quadlet). The MLB_MSS . LKSYSCMD bit is set by hardware and cleared by software.
0 (R/W0C)	RSTSYSCMD	Reset System Command Detected. The MLB_MSS . RSTSYSCMD bit indicates a reset system command was detected (in the system quadlet). The MLB_MSS . RSTSYSCMD bit is set by hardware, cleared by software.

21 Two-Wire Interface (TWI)

The processor has a two-wire interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I²C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

The TWI module offers the capabilities of simultaneous master and slave operation and support for both 7-bit addressing and multimedia data arbitration. The TWI interface uses two pins for transferring clock (TWI_SCL) and data (TWI_SDA) and supports the protocol at speeds up to 400K bits/sec. The TWI interface pins are compatible with 3.3V logic levels.

To preserve processor bandwidth, the TWI module can be set up with transfer-initiated interrupts to only service FIFO buffer data reads and writes. Protocol-related interrupts are optional. The TWI externally moves 8-bit data while maintaining compliance with the I²C bus protocol.

NOTE: All TWI instances (TWI0-5) operate in the SCLK0 domain.

TWI Features

The TWI is fully compatible with the widely used I²C bus standard.

The TWI controller includes the following features.

- Simultaneous master and slave operation on multiple device systems
- Support for multi-master bus arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support
- Master clock synchronization and support for clock low extension
- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate

- Individual override control of data and clock lines in the event of bus lock-up
- Input filter for spike suppression
- Serial camera control bus support as specified in the *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*
- Programmable drive/tolerance of TWI pins. Refer to the [PADS_PCFG0](#) register description in General-Purpose Ports (PORT) chapter for details.

TWI Functional Description

The TWI interface is a shift register that serially transmits and receives data bits. It moves data 1 bit at a time at the SCL rate, to and from other TWI devices. The SCL signal synchronizes the shifting and sampling of the data on the serial data pin.

ADSP-2156x TWI Register List

The Two-Wire Interface controller TWI allows a device to interface to an inter-IC bus as specified by the Philips I²C Bus Specification version 2.1, dated January 2000. A set of registers governs TWI operations. For more information on TWI functionality, see the TWI register descriptions.

Table 21-1: ADSP-2156x TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_IMSK	Interrupt Mask Register
TWI_ISTAT	Interrupt Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_RXDATA16	Rx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_SLVADDR	Slave Mode Address Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_TXDATA8	Tx Data Single-Byte Register

ADSP-2156x TWI Interrupt List

Table 21-2: ADSP-2156x TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
102	TWI0_DATA	TWI0 Data Interrupt	Level	
103	TWI1_DATA	TWI1 Data Interrupt	Level	
104	TWI2_DATA	TWI2 Data Interrupt	Level	
105	TWI3_DATA	TWI3 Data Interrupt	Level	
106	TWI4_DATA	TWI4 Data Interrupt	Level	
107	TWI5_DATA	TWI5 Data Interrupt	Level	

TWI Block Diagram

The *TWI Block Diagram* figure shows the basic blocks of the TWI interface.

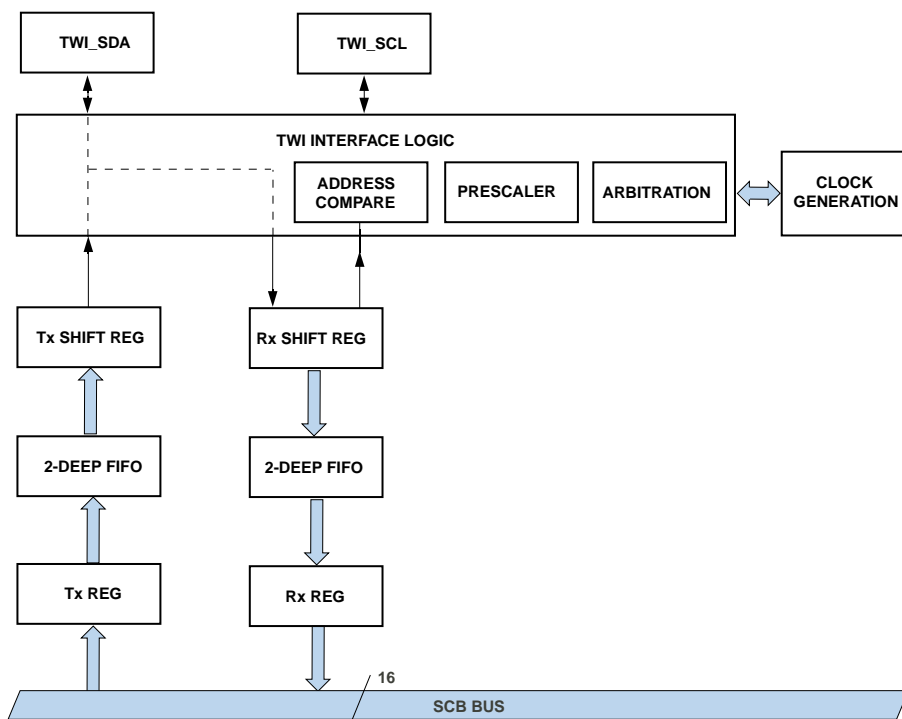


Figure 21-1: TWI Block Diagram

External Interface

The TWI_SDA (serial data) and TWI_SCL (serial clock) signals are open drain and require pull-up resistors. These bidirectional signals externally interface the TWI controller to the I²C bus and no other external connections or logic are necessary.

Serial Clock Signal (SCL)

The serial clock signal (`TWI_SCL`) is an input in slave mode. In master mode, the TWI controller must set this signal to the desired frequency.

The TWI controller supports the standard mode of operation (up to 100 kHz) or fast mode (up to 400 kHz). The TWI control register (`TWI_CTL`) sets the `TWI_CTL.PRESCALE` value which sets the relationship between the system clock (SCLK) and the internally timed events of the TWI controller. The internal time reference is derived from SCLK using a prescaled value. The prescale value is the number of SCLK periods used in the generation of one internal time reference. Set the value of prescale to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value as follows.

$$\text{PRESCALE} = f_{\text{SCLK}}/10\text{MHz}$$

NOTE: It is not always possible to achieve 10-MHz accuracy. In such cases, it is safe to round up the `PRESCALE` value to the next highest integer. For example, if SCLK is 100 MHz, the `PRESCALE` value is calculated as $100\text{ MHz}/10\text{ MHz} = 10$. A prescale value of 14 in this case ensures that all timing requirements are met.

During master mode operation, the TWI module uses the `TWI_CLKDIV` register values to create the minimum `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` durations of the `TWI_SCL` signal. The `TWI_CLKDIV.CLKHI` field specifies the minimum number of 10-MHz time reference periods the `TWI_SCL` waits before a new clock low period begins, assuming a single master. (The 10-MHz time reference periods are represented as an 8-bit binary value). The TWI uses the `TWI_CLKDIV.CLKLO` field to specify the minimum number of internal time reference periods (represented as an 8-bit binary value); the `TWI_SCL` signal is held low.

Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the clock generated is 1/10 MHz or 100 ns. The following equation describes the frequency.

$$\text{CLKDIV} = \text{TWI_SCL period}/10\text{ MHz time reference.}$$

For example, for an `TWI_SCL` of 400 kHz (period = $1/400\text{ kHz} = 2500\text{ ns}$) and an internal time reference of 10 MHz (period = 100 ns), the following equation applies:

$$\text{CLKDIV} = 2500\text{ ns}/100\text{ ns} = 25$$

Therefore, a `TWI_SCL` signal with a 30% duty cycle has `TWI_CLKDIV.CLKLO`=17 and `TWI_CLKDIV.CLKHI`=8. Adding `TWI_CLKDIV.CLKLO` and `TWI_CLKDIV.CLKHI` equals `CLKDIV`.

NOTE: The `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` fields are not intended to guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for the `TWI_SCL` signal. Slew rate controls falling edges. The *RC* time constant governs the rising edges. The pull-up resistor and the `TWI_SCL` capacitance form the time constant. See the “Register Descriptions” section for more details.

Serial Data Signal (SDA)

The TWI transmits and receives serial data, depending on the direction of the transfer, on the bidirectional serial data signal (SDA).

Internal Interface

The peripheral bus interface supports the transfer of 16-bit wide data. The processor uses the interface in the support of register and FIFO buffer reads and writes. The TWI internal interface is comprised of the blocks described as follows.

Register block. Contains all control and status bits and reflects what can be written or read as outlined by the programming model. Each function block updates their corresponding status bits.

FIFO buffer. Configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

Transmit shift register. Serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgments or it can be manually overwritten.

Receive shift register. Receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

Address compare block. Supports address comparison in the event the TWI controller module is accessed as a slave.

Prescaler block. Must be programmed to generate a 10-MHz time reference relative to the system clock. The block uses this time base for filtering of data and timing events specified by the electrical data sheet (See the Philips specification). The block uses the time base to generate the TWI_SCL clock as well.

Clock generation module. Generates an external TWI_SCL clock when in master mode. It includes the logic necessary for synchronization in a multi-master clock configuration and clock stretching when configured in slave mode.

NOTE: The TWI does not support DMA based operation.

TWI Architectural Concepts

The TWI controller follows the transfer protocol of the Philips I²C Bus specification version 2.1 dated January 2000.

NOTE: The TWI unit does not support DMA-based operation.

TWI Protocol

The *Data Transfer* figure shows a simple complete transfer.

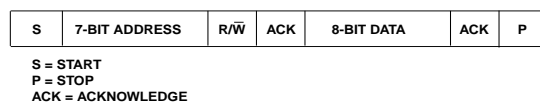


Figure 21-2: Data Transfer

The TWI controller register contents maps to a basic transfer. The *Data Transfer with Bit Illustration* figure details the same transfer from the *Data Transfer* figure noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits 1 byte of data. The slave has acknowledged both address and data.

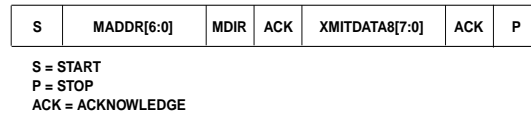


Figure 21-3: Data Transfer with Bit Illustration

Clock Generation and Synchronization

The TWI controller implementation only issues a clock during master mode operation and only at the time a transfer initiates. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. The *Clock Synchronization* figure shows this functionality.

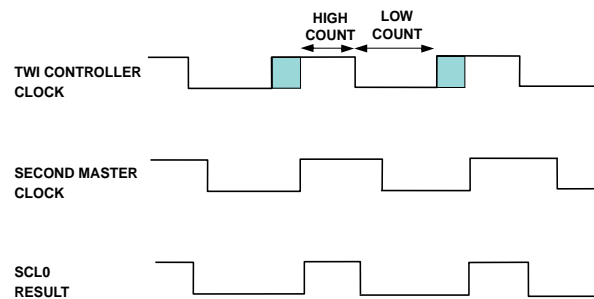


Figure 21-4: Clock Synchronization

The TWI controller serial clock (TWI_SCL) output follows these rules:

- Once the clock high (TWI_CLKDIV.CLKHI) count is complete, the serial clock output is driven low and the clock low (TWI_CLKDIV.CLKLO) count begins.
- Once the clock low count is complete, the serial clock line is three-stated. This state allows the external pull-up resistor to pull the TWI_SCL signal high. The clock synchronization logic enters into a delay mode (shaded area) until the TWI_SCL signal is detected at logic 1 level. Now, the clock high count begins.

Bus Arbitration

The TWI controller initiates a master mode transmission only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. The *Bus Arbitration* figure shows the arbitration.

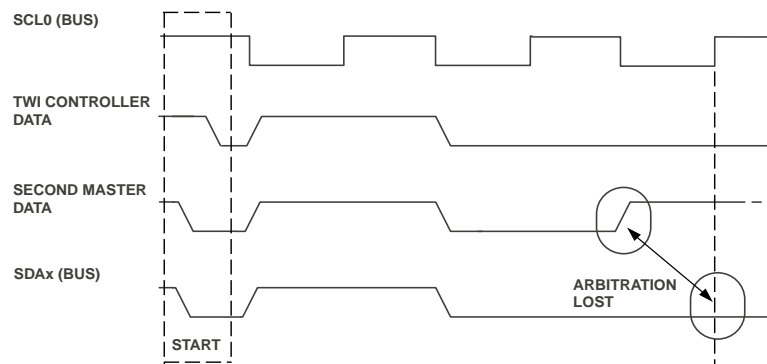


Figure 21-5: Bus Arbitration

The TWI controller monitors the serial data bus (SDA) while the `TWI_SCL` signal is high. If the `TWI_SDA` signal is determined to be an active logic 0 level while the data of the TWI controller is a logic 1 level, the TWI controller has lost arbitration. It stops generating the clock and data signals. Arbitration is not only performed at the serial clock edges, but also during the entire time the `TWI_SCL` signal is high.

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is a logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, except repeated start combined transfers. The *Start and Stop Conditions* figure shows the transitions.

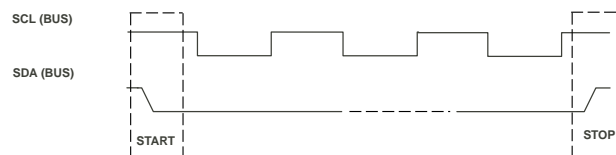


Figure 21-6: Start and Stop Conditions

The TWI special case start and stop conditions of the TWI controller include the following.

- Controller addressed as a slave-receiver. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_I_STAT . SCOMP`).
- Controller addressed as a slave-transmitter. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_I_STAT . SCOMP`) and indicates a slave transfer error (`TWI_I_STAT . SERR`).
- Controller as a master-transmitter or master-receiver. If the stop bit (`TWI_MSTRCTL . STOP`) is set during an active master transfer, the TWI controller issues a stop condition as soon as possible avoiding any error conditions. The TWI controller operates as if data transfer count had been reached.

General Call Support

The TWI controller always decodes and acknowledges a general call address if:

- The TWI controller is enabled as a slave

- General call is enabled

The `TWI_SLVCTL.GEN` bit configures general call addressing (0x00) only when the TWI controller is a slave-receiver.

If the data associated with the transfer is (NAK) not acknowledged, the `TWI_SLVCTL.NAK` bit can be set. If the TWI controller issues a general call as a master-transmitter, set the appropriate address (`TWI_MSTRADDR` register) and transfer direction (`TWI_MSTRCTL.DIR` bit) and load the transmit FIFO data.

NOTE: The byte following the general call address usually defines the slaves response to the call. The interpretation of the command in the second byte is based on the value of its LSB. For a TWI slave device, the bytes received after the general call address are considered data.

Fast Mode

Fast mode essentially uses the same mechanics as the standard mode of operation. Fast mode affects electrical specifications and timing. When fast mode is enabled, (FAST) timing is modified to meet the following electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition set-up time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

TWI Operating Modes

The TWI has two modes of operation: repeated start and clock stretching. The following sections describe the operating modes.

Repeated Start

A repeated start condition is the absence of a stop condition between two transfers. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. The following sections guide the programmer in developing a service routine.

Transmit Receive Repeated Start

The *Repeated Start Followed by Data Receive* figure shows a repeated start followed by a data receive sequence. The shading in the figure indicates that the slave has control of the bus.

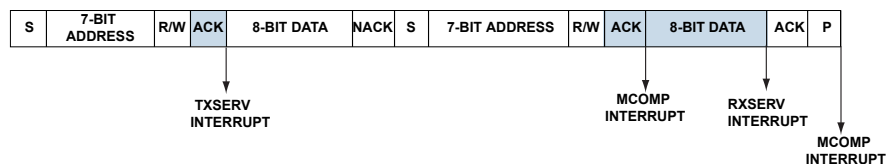


Figure 21-7: Repeated Start Followed by Data Receive

The tasks performed at each interrupt are:

- Transmit FIFO service (`TWI_I_STAT.TXSERV`) interrupt request. This interrupt is generated due to a FIFO access. Since this byte is the last of this transfer, the TWI uses the `TWI_FIFOSTAT` register to indicate that the transmit FIFO is empty. When read, `TWI_MSTRCTL.DCNT` bit field=0. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and set the `TWI_MSTRCTL.DIR` bit if the following transfer is a data receive.
- Master transfer complete (`TWI_I_STAT.MCOMP`) interrupt. This interrupt request is generated when all data transfers (`TWI_MSTRCTL.DCNT` bit field=0). If no errors occur, a start condition initiates. Clear the `TWI_MSTRCTL.RSTART` bit and program the `TWI_MSTRCTL.DCNT` bits with the desired number of bytes to receive.
- Receive FIFO service (`TWI_I_STAT.RXSERV`) interrupt. This interrupt request is generated due to the arrival of a byte in the receive FIFO. Simple data handling is the only requirement.
- Master transfer complete (`TWI_I_STAT.MCOMP`) interrupt. This interrupt request is generated when the transfer completes.

Receive Transmit Repeated Start

The *Repeated Start Data Receive Followed by Data Transmit* figure illustrates a repeated start data receive followed by a data transmit sequence. The shading in the figure indicates that the slave has control of the bus.

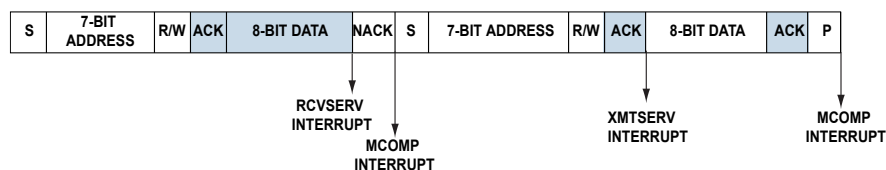


Figure 21-8: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (`TWI_I_STAT.RXSERV`) interrupt. This interrupt request is generated due to the arrival of a data byte in the receive FIFO. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and clear the `TWI_MSTRCTL.DIR` bit if the following transfer is a data transmit.
- Master transfer complete (`TWI_I_STAT.MCOMP`) interrupt. This interrupt request has occurred due to the completion of the data receive transfer. If no errors occur, a start condition initiates. Clear the `TWI_MSTRCTL.RSTART` bit and program the `TWI_MSTRCTL.DCNT` bits with the desired number of bytes to transmit.
- Transmit FIFO service (`TWI_I_STAT.TXSERV`) interrupt. This interrupt request is generated due to a FIFO access. Simple data handling is the only requirement.
- Master transfer complete (`TWI_I_STAT.MCOMP`) interrupt. This interrupt request is generated when the transfer completes.

NOTE: There is no timing constraint to meet the conditions—program the bits as required. Refer to [Clock Stretching During Repeated Start](#) section for more on how the controller stretches the clock during repeated start transfers.

Clock Stretching

Clock stretching is an added function of the TWI controller in master mode operation. This behavior uses self-induced stretching of the I²C clock while waiting to service interrupts. Hardware initiates stretching automatically. No programming is necessary. The TWI controller as a master supports three modes of clock stretching:

- [Clock Stretching During FIFO Underflow](#)
- [Clock Stretching During FIFO Overflow](#)
- [Clock Stretching During Repeated Start](#)

Clock Stretching During FIFO Underflow

During a master mode transmit, an interrupt request occurs the instant the transmit FIFO becomes empty. The most recent byte begins transmission. If the `TWI_I_STAT.TXSERV` interrupt request is not serviced, the concluding acknowledge phase of the transfer stretches.

Stretching of the clock continues until new data bytes are written to the transmit FIFO (`TWI_TXDATA8` or `TWI_TXDATA16` registers). No other action is required to release the clock and continue the transmission. This behavior continues until the transmission completes (`TWI_MSTRCTL.DCNT=0`). The transmission concludes (`TWI_I_STAT.MCOMP`). The *Clock Stretching during FIFO Underflow* figure and table show the stretching.

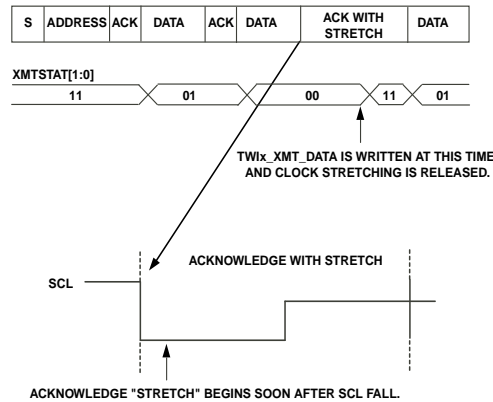


Figure 21-9: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear the interrupt request source bits.

Clock Stretching During FIFO Overflow

During a master mode receive operation, an interrupt occurs at the instant the receive FIFO becomes full. It is during the acknowledge phase of this received byte that clock stretching begins. The TWI module makes no attempt to initiate the reception of another byte. Stretching of the clock continues until the data bytes previously received are read from the receive FIFO buffer (`TWI_RXDATA8` or `TWI_RXDATA16` registers). No other action is required to release the clock and continue the reception of data. This behavior continues until the reception is complete (`TWI_MSTRCTL.DCNT=0`). Reception concludes (`TWI_ISTAT.MCOMP`). The *Clock Stretching During FIFO Overflow* figure and table show the clock stretching.

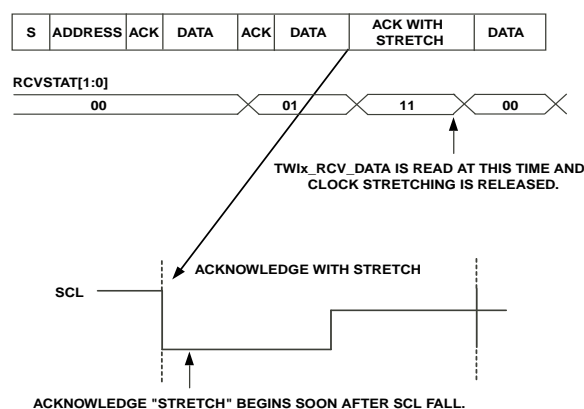


Figure 21-10: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Acknowledge: Clear the interrupt source bits.	Interrupt: MCOMP – Master receive complete.

Clock Stretching During Repeated Start

The repeated start feature in I²C protocol requires a transition between two subsequent transfers. With the use of clock stretching, the task of managing transitions becomes simpler and common to all transfer types.

Once an initial TWI master transfer completes (transmit or receive), the clock initiates a stretch during the repeated start phase between transfers. Concurrent with this event, the initial transfer generates a `TWI_ISTAT.MCOMP` interrupt to signify the initial transfer has completed (`TWI_MSTRCTL.DCNT=0`). This initial transfer is handled without any special bit setting sequences or timing.

The clock stretching logic described applies here. With no system-related timing constraints, the subsequent transfer (receive or transmit) is set up and activated. This sequence can repeat as many times as required to string a series of repeated start transfers together. The *Clock Stretching during Repeated Start Condition* figure and table show the clock stretching.

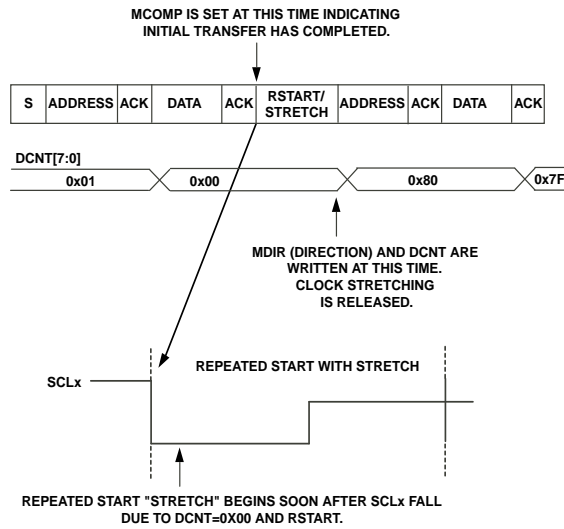


Figure 21-11: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear the interrupt request source bits. Write to TWIx_MASTER_CTL, setting MDIR (receive), clearing RSTART, and setting new DCNT value (nonzero).
Interrupt: RCVSERV – Receive FIFO is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear the interrupt request source bits.

TWI Programming Model

The topics in this section provide information on the basic programming steps required to set up and run the two wire interface.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operations.

Perform general setup before setting either the master or slave enable bits.

1. Program the `TWI_CTL.EN` bit to enable the TWI controller and set the prescale value (`TWI_CTL.PRESCALE` bit).
2. Program the prescale value to the binary representation of $f_{SCLK0}/10$ MHz. Round up all values to the next whole number.
3. Set the `TWI_CTL.EN` bit to enable the controller.

Once the TWI controller is enabled, a bus busy condition can be detected. This condition clears after t_{BUF} has expired, assuming no additional bus activity has been detected.

Slave Mode

When enabled, slave mode operation supports both receive and transmit data transfers.

It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. The following setup reflects this functionality.

1. Program the `TWI_SLVADDR` register. The TWI uses the appropriate 7 bits in determining a match during the address phase of the transfer.
2. Program the `TWI_TXDATA8.VALUE` or `TWI_TXDATA16` registers. These values are the initial data values for transmission when the slave is addressed and transmission is needed. This step is optional. If no data is written when the slave is addressed and transmission is needed, the serial clock (`TWI_SCL`) stretches. An interrupt is generated until data is written to the transmit FIFO.
3. Program the `TWI_IMSK` register. There are enable-bits associated with the desired interrupt sources. For example, programming the value `0x000F` results in an interrupt request output to the processor, when the TWI module detects a valid address match. An interrupt request also occurs when a valid slave transfer completes or has an error, or a subsequent transfer has begun and the previous transfer has not been serviced.
4. Program the `TWI_SLVCTL` register. This step prepares and enables slave mode operation. For example, programming the value `0x0005` enables slave mode operation and requires 7-bit addressing. It indicates that data in the transmit FIFO buffer is for slave mode transmission.

The *Slave Mode Interaction* table and *TWI Slave Mode Program Flow* diagram represent the interaction between the TWI controller and the processor using this example.

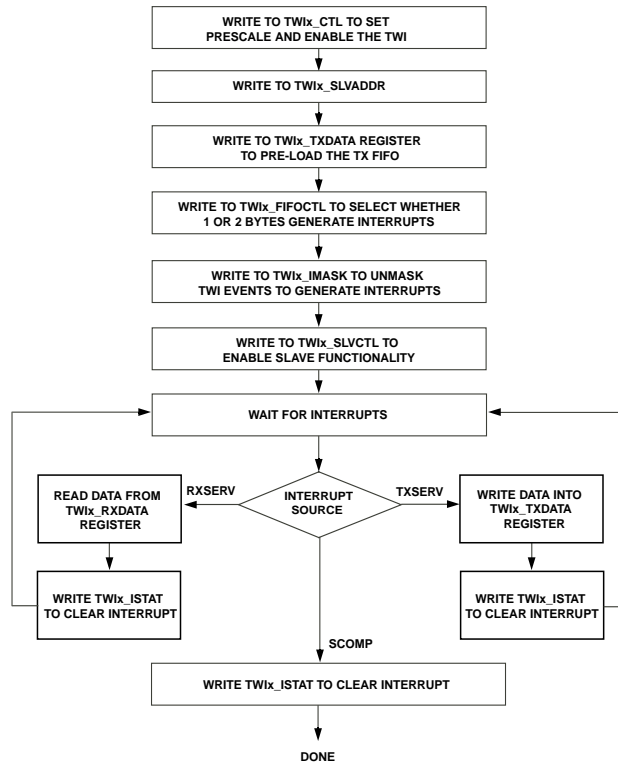


Figure 21-12: TWI Slave Mode Program Flow

Table 21-3: Slave Mode Interaction

TWI Controller	Processor
Interrupt: SINIT – Slave transfer in progress.	Acknowledge: Clear the interrupt source bits.
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read TWIx_FIFO_STAT. Read the receive FIFO buffer.
...	...
Interrupt: SCOMP – Slave transfer complete.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

Master Mode Program Flow

The *Master Mode Program Flow* figure shows the program for the TWI in master mode.

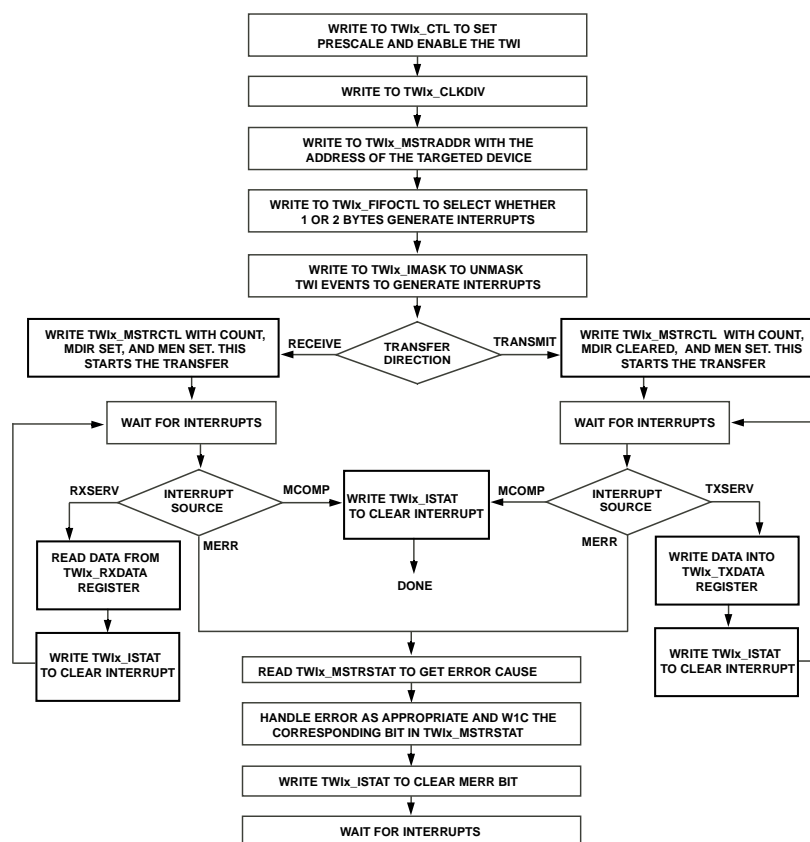


Figure 21-13: Master Mode Program Flow

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis.

An example of programming steps for a receive and for a transmit is given separately in following sections. The programming step for clock setup listed here is common to both transfer types.

1. Program the `TWI_CLKDIV` register to define the minimum high and minimum low duration for the clock.

The `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` fields do not guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for `TWI_SCL`. The slew rate controls falling edges. The RC time constant formed by the pull-up resistor and the SCL capacitance govern rising edges. See the “Register Descriptions” section for more details.

Master Mode Transmit

Follow these programming steps for a single master mode transmission:

1. Program the `TWI_MSTRADDR` register. This step defines the address transmitted during the address phase of the transfer.

2. Program the `TWI_TXDATA8` or `TWI_TXDATA16` register. This step configures the initial data transmitted. It is an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.
3. Program the `TWI_FIFOCTL` register. The programming indicates if the transmit FIFO buffer interrupt requests occur with each byte transmitted (8-bits) or with every 2 bytes transmitted (16-bits).
4. Program the `TWI_IMSK` register. This step enables the bits associated with the desired interrupt request sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, and the master transfer has an error.
5. Program the `TWI_MSTRCTL` register. This step prepares and enables master mode operation. For example, programming the value 0x0201: enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

The *Master Mode Transmit Setup Interaction* table represents the interaction between the TWI controller and the processor using this example.

Table 21-4: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits.

Master Mode Receive

Follow these programming steps for a single master mode receive.

1. Program the `TWI_MSTRADDR` register. This step defines the address transmitted during the address phase of the transfer.
2. Program the `TWI_FIFOCTL` register. This step indicates if the receive FIFO buffer interrupt requests occur with each byte received (8-bits) or with every 2 bytes received (16-bits).
3. Program the `TWI_IMSK` register. This step configures the enable bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt request output to the processor when the master transfer completes, and the master transfer has an error.
4. Program the `TWI_MSTRCTL` register. This step prepares and enables master mode operation. For example, programming the value 0x0205: enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

The *Master Mode Receive Setup Interaction* table shows the interaction between the TWI controller and the processor using this example.

Table 21-5: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.

NOTE: After the `TWI_MSTRCTL.DCNT` bit decrements to zero, the TWI master device sends a NAK to indicate to the slave transmitter to release the bus. This operation allows the master to send the stop signal to terminate the transfer.

ADSP-2156x TWI Register Descriptions

Two-Wire Interface (TWI) contains the following registers.

Table 21-6: ADSP-2156x TWI Register List

Name	Description
<code>TWI_CLKDIV</code>	SCL Clock Divider Register
<code>TWI_CTL</code>	Control Register
<code>TWI_FIFOCTL</code>	FIFO Control Register
<code>TWI_FIFOSTAT</code>	FIFO Status Register
<code>TWI_IMSK</code>	Interrupt Mask Register
<code>TWI_ISTAT</code>	Interrupt Status Register
<code>TWI_MSTRADDR</code>	Master Mode Address Register
<code>TWI_MSTRCTL</code>	Master Mode Control Registers
<code>TWI_MSTRSTAT</code>	Master Mode Status Register
<code>TWI_RXDATA16</code>	Rx Data Double-Byte Register
<code>TWI_RXDATA8</code>	Rx Data Single-Byte Register
<code>TWI_SLVADDR</code>	Slave Mode Address Register
<code>TWI_SLVCTL</code>	Slave Mode Control Register
<code>TWI_SLVSTAT</code>	Slave Mode Status Register
<code>TWI_TXDATA16</code>	Tx Data Double-Byte Register
<code>TWI_TXDATA8</code>	Tx Data Single-Byte Register

SCL Clock Divider Register

During master mode operation, the `TWI_CLKDIV` holds values, which the TWI uses to create the high and low durations of the serial clock (SCL). The clock signal SCL is an output in master mode and an input in slave mode. The values in the `TWI_CLKDIV.CLKLO` and `TWI_CLKDIV.CLKHI` fields add up to the `CLKDIV` value the following equation.

$$\text{CLKDIV} = \text{TWI SCL period} / 10 \text{ MHz time reference}$$

Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns. For example, for an SCL of 400 KHz (period = 1/400 KHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

$$\text{CLKDIV} = 2500 \text{ ns} / 100 \text{ ns} = 25$$

For an SCL with a 30% duty cycle, use `TWI_CLKDIV.CLKLO = 17` and `TWI_CLKDIV.CLKHI = 8`.

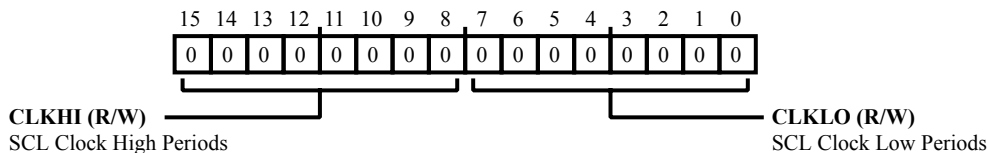


Figure 21-14: TWI_CLKDIV Register Diagram

Table 21-7: TWI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	CLKHI	SCL Clock High Periods. The <code>TWI_CLKDIV.CLKHI</code> specifies the number of 10 MHz time reference periods the serial clock (SCL) waits before a new clock low period begins, assuming a single master.
7:0 (R/W)	CLKLO	SCL Clock Low Periods. The <code>TWI_CLKDIV.CLKLO</code> specifies the number of internal time reference periods the serial clock (SCL) is held low.

Control Register

The `TWI_CTL` enables the TWI, establishes a relationship between the system clock (`SCLK`) and the TWI controller's internally timed events, and enables SCCB compatibility.

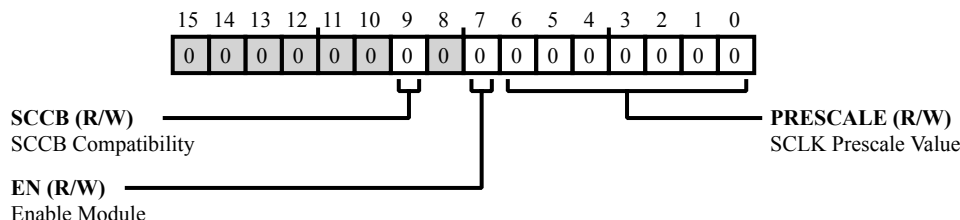


Figure 21-15: TWI_CTL Register Diagram

Table 21-8: TWI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	SCCB	SCCB Compatibility. The <code>TWI_CTL.SCCB</code> enables SCCB compatible operation for the TWI. SCCB compatibility is an optional feature and should not be used in an I ² C bus system. When this feature is enabled, all slave asserted acknowledgement bits are ignored by this master. This feature is valid only during transfers where the TWI is mastering an SCCB bus. Slave mode transfers should be avoided when this feature is enabled because the TWI controller always generates an acknowledge in slave mode.
		0 Disable SCCB compatibility. When disabled, master transfers are not SCCB compatible.
		1 Enable SCCB compatibility. When enabled, master transfers are SCCB compatible. All slave-asserted acknowledgement bits are ignored by this master.
7 (R/W)	EN	Enable Module. The <code>TWI_CTL.EN</code> enables TWI controller operation for either master and/or slave mode of operation. It is recommended that this bit be set at the time <code>TWI_CTL.PRESCALE</code> is initialized and remain set. This method guarantees accurate operation of bus busy detection logic.
		0 Disable
		1 Enable

Table 21-8: TWI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	PRESCALE	<p>SCLK Prescale Value.</p> <p>The <code>TWI_CTL.PRESCALE</code> holds the pre-scaled value for the TWI internal time reference. This reference is derived from SCLK according to the formula:</p> $\text{TWI_CTL.PRESCALE} = f_{\text{SCLK}}/10\text{MHz}$ <p>The <code>TWI_CTL.PRESCALE</code> specifies the number of system clock (SCLK) periods used in the generation of one internal time reference. The value of <code>TWI_CTL.PRESCALE</code> must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value.</p>

FIFO Control Register

The `TWI_FIFOCTL` control bits affect only the FIFO and are not tied in any way with master or slave mode operation.

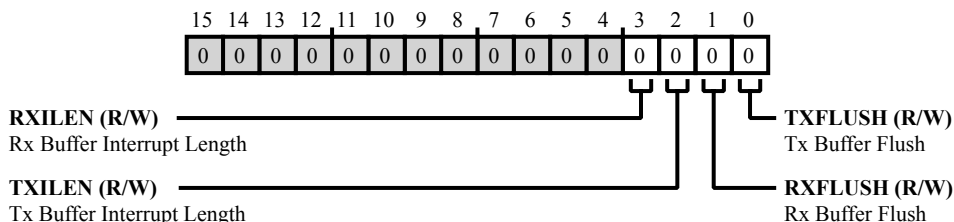


Figure 21-16: TWI_FIFOCTL Register Diagram

Table 21-9: TWI_FIFOCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	RXILEN	Rx Buffer Interrupt Length. The <code>TWI_FIFOCTL.RXILEN</code> determines the rate at which receive buffer interrupts are to be generated. Interrupts may be generated with each byte received or after two bytes are received. Interrupt status is available in <code>TWI_FIFOSTAT.RXSTAT</code> .
		0 RXSERVI on 1 or 2 Bytes in FIFO
		1 RXSERVI on 2 Bytes in FIFO
2 (R/W)	TXILEN	Tx Buffer Interrupt Length. The <code>TWI_FIFOCTL.TXILEN</code> determines the rate at which transmit buffer interrupts are to be generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. Interrupt status is available in <code>TWI_FIFOSTAT.TXSTAT</code> .
		0 TXSERVI on 1 Byte of FIFO Empty
		1 TXSERVI on 2 Bytes of FIFO Empty
1 (R/W)	RXFLUSH	Rx Buffer Flush. The <code>TWI_FIFOCTL.RXFLUSH</code> directs the TWI to flush the contents of the receive buffer and update <code>TWI_FIFOSTAT.RXSTAT</code> to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive, the receive buffer in this state responds to the receive logic as if it is full.
		0 Normal Operation of Rx Buffer
		1 Flush Rx Buffer

Table 21-9: TWI_FIFOCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	TXFLUSH	Tx Buffer Flush. The <code>TWI_FIFOCTL.TXFLUSH</code> directs the TWI to flush the contents of the transmit buffer and update <code>TWI_FIFOSTAT.TXSTAT</code> to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds to the transmit logic as if it is empty.	
		0	Normal Operation of Tx Buffer
		1	Flush Tx Buffer

FIFO Status Register

The `TWI_FIFOSTAT` fields indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

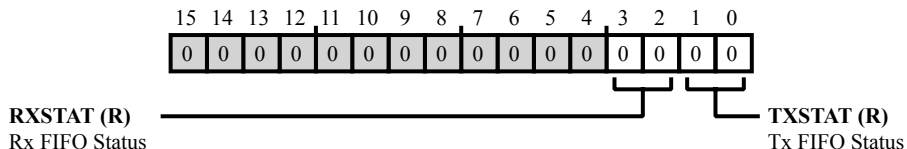


Figure 21-17: TWI_FIFOSTAT Register Diagram

Table 21-10: TWI_FIFOSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:2 (R/NW)	RXSTAT	Rx FIFO Status. The read-only <code>TWI_FIFOSTAT.RXSTAT</code> indicates the number of valid data bytes in the receive FIFO buffer. The status is updated with each FIFO buffer read using the peripheral data bus or write access by the receive shift register. Simultaneous accesses are allowed.
		0 Empty. The FIFO is empty.
		1 Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral read of the FIFO is allowed.
		2 Reserved
		3 Full. The FIFO is full and contains two bytes of data. Either a single or double byte peripheral read of the FIFO is allowed.
1:0 (R/NW)	TXSTAT	Tx FIFO Status. The read-only <code>TWI_FIFOSTAT.TXSTAT</code> field indicates the number of valid data bytes in the FIFO buffer. The status is updated with each FIFO buffer write using the peripheral data bus or read access by the transmit shift register. Simultaneous accesses are allowed.
		0 Empty. The FIFO is empty. Either a single or double byte peripheral write of the FIFO is allowed.
		1 Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral write of the FIFO is allowed.
		2 Reserved
		3 Full. The FIFO is full and contains two bytes of data.

Interrupt Mask Register

The `TWI_IMSK` enables interrupt sources to assert the interrupt output. Each mask bit corresponds with one interrupt request source bit in `TWI_ISTAT`. Reading and writing `TWI_IMSK` does not affect the contents of the `TWI_ISTAT`.

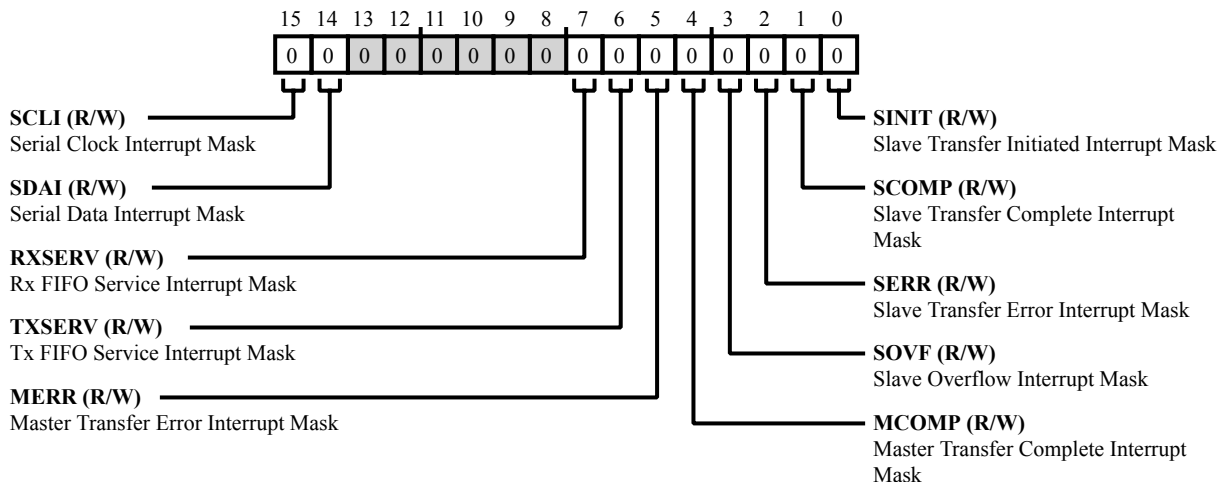


Figure 21-18: TWI_IMSK Register Diagram

Table 21-11: TWI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	SCLI	Serial Clock Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
14 (R/W)	SDAI	Serial Data Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
7 (R/W)	RXSERV	Rx FIFO Service Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
6 (R/W)	TXSERV	Tx FIFO Service Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt

Table 21-11: TWI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	MERR	Master Transfer Error Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
4 (R/W)	MCOMP	Master Transfer Complete Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
3 (R/W)	SOVF	Slave Overflow Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
2 (R/W)	SERR	Slave Transfer Error Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
1 (R/W)	SCOMP	Slave Transfer Complete Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt
0 (R/W)	SINIT	Slave Transfer Initiated Interrupt Mask.
		0 Mask (Disable) Interrupt
		1 Unmask (Enable) Interrupt

Interrupt Status Register

The `TWI_ISTAT` contains information about functional areas requiring servicing. Many of the bits serve as an indicator to further read and service various status registers. After servicing the interrupt source associated with a bit, the user must clear that interrupt source bit by writing a 1 to it.

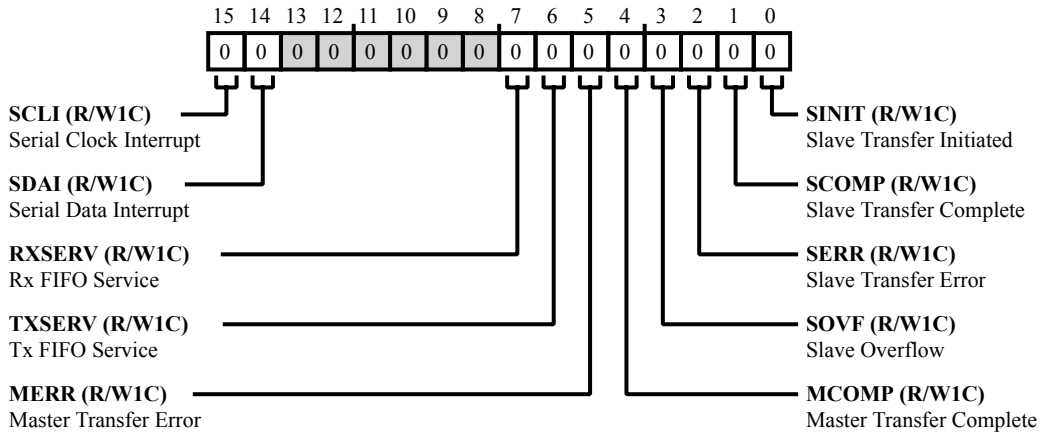


Figure 21-19: TWI_ISTAT Register Diagram

Table 21-12: TWI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	SCLI	Serial Clock Interrupt. If the TWI is enabled (<code>TWI_CTL.EN</code>), SCLI is set on a high-to-low transition of the serial clock pin (<code>SCLx</code>). Normally, this bit is not required for I ² C bus transfers. It will be initially set on an I ² C transfer and does not require clearing.
		0 No Interrupt. No transition was detected on the <code>SCLx</code> pin.
		1 Interrupt Detected. A high-to-low transition was detected on the <code>SCLx</code> pin. This bit is W1C.
14 (R/W1C)	SDAI	Serial Data Interrupt. If the TWI is enabled (<code>TWI_CTL.EN</code>), SDAI is set on a high-to-low transition of the serial data pin (<code>SDAx</code>). Normally, this bit is not required for I ² C bus transfers. It will be initially set on an I ² C transfer and does not require clearing.
		0 No Interrupt. No transition was detected on the <code>SDAx</code> pin.
		1 Interrupt Detected. A high-to-low transition was detected on the <code>SDAx</code> pin. This bit is W1C.

Table 21-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W1C)	RXSERV	Rx FIFO Service. If <code>TWI_FIFOCTL.RXILEN = 0</code> , the <code>TWI_ISTAT.RXSERV</code> is set each time the <code>TWI_FIFOSTAT.RXSTAT</code> field is updated to either 01 or 11. If <code>TWI_FIFOCTL.RXILEN = 1</code> , the <code>TWI_ISTAT.RXSERV</code> is set each time <code>TWI_FIFOSTAT.RXSTAT</code> is updated to 11.
		0 No Interrupt. The FIFO does not require servicing, or the <code>TWI_FIFOSTAT.RXSTAT</code> field has not changed since this bit was last cleared.
		1 Interrupt Detected. The receive FIFO buffer has one or two 8-bit words of data available to be read.
6 (R/W1C)	TXSERV	Tx FIFO Service. If <code>TWI_FIFOCTL.TXILEN = 0</code> , the <code>TWI_ISTAT.TXSERV</code> is set each time the <code>TWI_FIFOSTAT.TXSTAT</code> field is updated to either 01 or 00. If <code>TWI_FIFOCTL.TXILEN = 1</code> , the <code>TWI_ISTAT.TXSERV</code> is set each time <code>TWI_FIFOSTAT.TXSTAT</code> is updated to 00.
		0 No Interrupt. FIFO does not require servicing, or the <code>TWI_FIFOSTAT.TXSTAT</code> field has not changed since this bit was last cleared.
		1 Interrupt Detected. The transmit FIFO buffer has one or two 8-bit locations available to be written.
5 (R/W1C)	MERR	Master Transfer Error. The <code>TWI_ISTAT.MERR</code> indicates that a master error has occurred. The conditions surrounding the error are indicated by the master status register (<code>TWI_MSTRSTAT</code>).
		0 No Interrupt
		1 Interrupt Detected
4 (R/W1C)	MCOMP	Master Transfer Complete. The <code>TWI_ISTAT.MCOMP</code> indicates that the initiated master transfer has completed. In the absence of a repeat start, the bus has been released.
		0 No Interrupt
		1 Interrupt Detected
3 (R/W1C)	SOVF	Slave Overflow. The <code>TWI_ISTAT.SOVF</code> indicates that the <code>TWI_ISTAT.SCOMP</code> bit was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another.
		0 No Interrupt
		1 Interrupt Detected

Table 21-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	SERR	Slave Transfer Error. The <code>TWI_ISTAT.SERR</code> indicates that a slave error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.
		0 No Interrupt
		1 Interrupt Detected
1 (R/W1C)	SCOMP	Slave Transfer Complete. The <code>TWI_ISTAT.SCOMP</code> indicates that the transfer is complete and either a stop, or a restart was detected.
		0 No Interrupt
		1 Interrupt Detected
0 (R/W1C)	SINIT	Slave Transfer Initiated. The <code>TWI_ISTAT.SINIT</code> indicates whether or not a slave transfer is in progress.
		0 No Interrupt. A transfer is not in progress, or an address match has not occurred since the last time this bit was cleared.
		1 Interrupt Detected. The slave has detected an address match, and a transfer has been initiated.

Master Mode Address Register

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of `TWI_MSTRADDR`. When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is `b#1010000X`, where `X` is the read/write bit, the `TWI_MSTRADDR` is programmed with `b#1010000`, which corresponds to `0x50`. When sending out the address on the bus, the TWI controller appends the read/write bit as appropriate based on the state of the `TWI_MSTRCTL.DIR` bit.

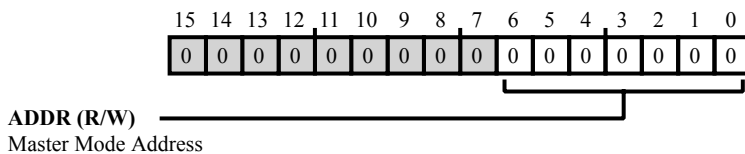


Figure 21-20: TWI_MSTRADDR Register Diagram

Table 21-13: TWI_MSTRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Master Mode Address.

Master Mode Control Registers

The `TWI_MSTRCTL` controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

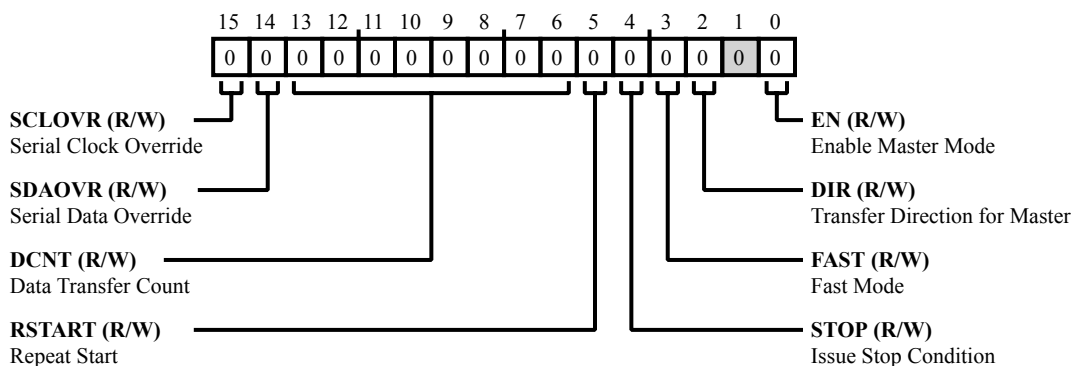


Figure 21-21: TWI_MSTRCTL Register Diagram

Table 21-14: TWI_MSTRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	SCLOVR	Serial Clock Override. The <code>TWI_MSTRCTL.SCLOVR</code> provides direct control of the serial clock line when required. Normal master and slave mode operation should not require override operation. When <code>TWI_MSTRCTL.SCLOVR</code> is set, the TWI overrides normal serial clock output, driving it to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When <code>TWI_MSTRCTL.SCLOVR</code> is cleared, the TWI permits normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic.
		0 Permit Normal SCL Operation
		1 Override Normal SCL Operation
14 (R/W)	SDAOVR	Serial Data Override. The <code>TWI_MSTRCTL.SDAOVR</code> provides direct control of the serial data line when required. Normal master and slave mode operation should not require override operation. When <code>TWI_MSTRCTL.SDAOVR</code> is set, the TWI overrides normal serial data operation under the control of the transmit shift register and acknowledge logic, driving serial data output to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When <code>TWI_MSTRCTL.SDAOVR</code> is cleared, the TWI permits normal serial data operation.
		0 Permit Normal SDA Operation
		1 Override Normal SDA Operation

Table 21-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:6 (R/W)	DCNT	<p>Data Transfer Count.</p> <p>The <code>TWI_MSTRCTL.DCNT</code> indicates the number of data bytes to transfer. As each data word is transferred, the TWI decrements this counter. When <code>TWI_MSTRCTL.DCNT</code> decrements to 0, a stop condition is generated. Setting <code>TWI_MSTRCTL.DCNT</code> to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the <code>TWI_MSTRCTL.STOP</code> bit. In the event a master transmit is aborted due to a slave data NAK, the value of <code>TWI_MSTRCTL.DCNT</code> equals the number of bytes not sent. The byte which was NAK'ed by the slave is counted as a sent byte.</p>
5 (R/W)	RSTART	<p>Repeat Start.</p> <p>The <code>TWI_MSTRCTL.RSTART</code> enables the TWI to issue a repeat start condition at the conclusion of the current transfer (<code>TWI_MSTRCTL.DCNT = 0</code>) and begin the next transfer. The current transfer concludes with updates to the appropriate status and interrupt bits. If errors occurred during the previous transfer, a repeat start does not occur. In the absence of any errors, master enable (<code>TWI_MSTRCTL.EN</code>) does not self clear on a repeat start.</p>
		0 Disable Repeat Start
		1 Enable Repeat Start
4 (R/W)	STOP	<p>Issue Stop Condition.</p> <p>The <code>TWI_MSTRCTL.STOP</code> directs the TWI to issue a stop condition. The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached). At that time, the <code>TWI_IMSK</code> is updated along with any associated status bits.</p>
		0 Permit Normal Operation
		1 Issue Stop
3 (R/W)	FAST	<p>Fast Mode.</p> <p>The <code>TWI_MSTRCTL.FAST</code> selects whether the TWI operates in fast mode or standard mode. In fast mode, the TWI uses timing specifications for transfers at up to 400K bits/s. In standard mode, the TWI uses timing specifications for transfers at up to 100K bits/s.</p>
		0 Select Standard Mode
		1 Select Fast Mode
2 (R/W)	DIR	<p>Transfer Direction for Master.</p> <p>The <code>TWI_MSTRCTL.DIR</code> selects the transfer direction for the TWI as master initiated receive or transmit.</p>
		0 Master Transmit
		1 Master Receive

Table 21-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration		
0 (R/W)	EN	<p>Enable Master Mode.</p> <p>The <code>TWI_MSTRCTL.EN</code> enables master mode functionality. A start condition is generated if the bus is idle. This bit self clears at the completion of a transfer (after <code>TWI_MSTRCTL.DCNT</code> decrements to zero), including transfers terminated due to errors.</p> <p>If disabled (=0) during operation, the transfer is aborted, and all logic associated with master mode transfers are reset. Serial data and serial clock (SDA, SCL) are no longer driven. Write-1-to-clear status bits are not affected.</p>		
		<table border="1"> <tr> <td data-bbox="932 699 932 743">0</td> <td data-bbox="932 699 1528 743">Disable</td> </tr> </table>	0	Disable
0	Disable			
		<table border="1"> <tr> <td data-bbox="932 751 932 795">1</td> <td data-bbox="932 751 1528 795">Enable</td> </tr> </table>	1	Enable
1	Enable			

Master Mode Status Register

The `TWI_MSTRSTAT` holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupt requests, but these bits offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Note that while `TWI_MSTRSTAT.SCLSEN` is set (this condition could be due to having no pull-up resistor on `TWI_SCL` or another agent is driving `TWI_SCL` low), the acknowledge bits (`TWI_MSTRSTAT.ANAK` and `TWI_MSTRSTAT.DNAK`) do not update. This result occurs because the acknowledge conditions are sampled during the high phase of `TWI_SCL`.

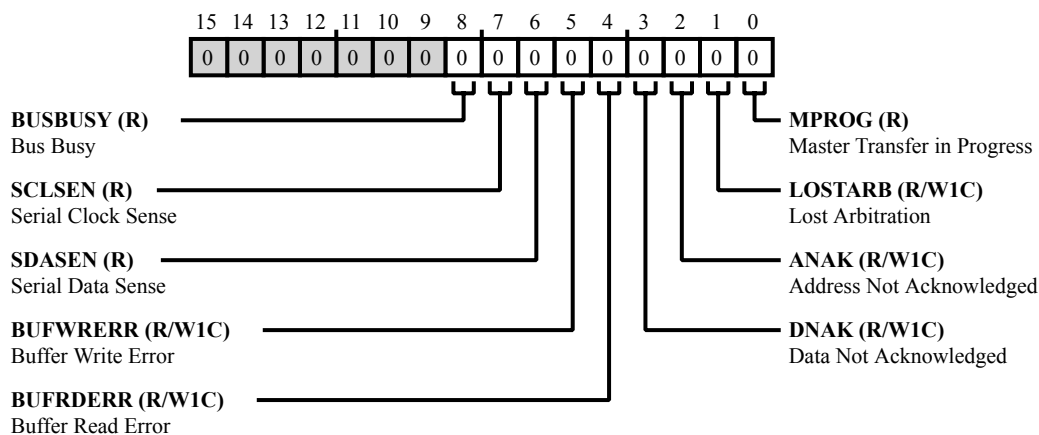


Figure 21-22: TWI_MSTRSTAT Register Diagram

Table 21-15: TWI_MSTRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/NW)	BUSBUSY	Bus Busy. The <code>TWI_MSTRSTAT.BUSBUSY</code> indicates whether the bus is currently busy or free. This indication is not limited to only this device but is for all devices. On a start condition, the setting of the register value is delayed due to the input filtering. On a stop condition the clearing of the register value occurs after t_{BUF} .
		0 Bus Free. The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time.
		1 Bus Busy. The bus is busy. Clock or data activity has been detected.

Table 21-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	SCLSEN	Serial Clock Sense. The <code>TWI_MSTRSTAT.SCLSEN</code> indicates the active or inactive state of the serial clock. Use this status bit when direct sensing of the serial clock line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.
		0 SCL Inactive "One". An inactive "one" is being sensed on the serial clock.
		1 SCL Active "Zero". An active "zero" is being sensed on the serial clock. The source of the active driver is not known and can be internal or external.
6 (R/NW)	SDASEN	Serial Data Sense. The <code>TWI_MSTRSTAT.SDASEN</code> indicates the active or inactive status of the serial data. Use this status bit when direct sensing of the serial data line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.
		0 SDA Inactive "One". An inactive "one" is currently being sensed on the serial data line.
		1 SDA Active "Zero". An active "zero" is currently being sensed on the serial data line. The source of the active driver is not known and can be internal or external.
5 (R/W1C)	BUFWRERR	Buffer Write Error. The <code>TWI_MSTRSTAT.BUFWRERR</code> indicates whether the current master transfer was aborted due to a receive buffer write error. The receive buffer and receive shift register were both full at the same time. This bit is W1C.
		0 No Status
		1 Buffer Write Error
4 (R/W1C)	BUFRDERR	Buffer Read Error. The <code>TWI_MSTRSTAT.BUFRDERR</code> indicates whether the current master transfer was aborted due to a transmit buffer read error. The error occurs if the buffer is empty when data is required by the transmit shift register. This bit is W1C.
		0 No Status
		1 Buffer Read Error

Table 21-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1C)	DNAK	Data Not Acknowledged. The <code>TWI_MSTRSTAT.DNAK</code> indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.
		0 No Status
		1 Data NAK
2 (R/W1C)	ANAK	Address Not Acknowledged. The <code>TWI_MSTRSTAT.ANAK</code> indicates whether the current master transfer was aborted due to the detection of a NAK during the address phase of the transfer. This bit is W1C.
		0 No Status
		1 Address NAK
1 (R/W1C)	LOSTARB	Lost Arbitration. The <code>TWI_MSTRSTAT.LOSTARB</code> indicates whether the current transfer was aborted due to the loss of arbitration with another master. This bit is W1C.
		0 No Status
		1 Lost Arbitration
0 (R/NW)	MPROG	Master Transfer in Progress. The <code>TWI_MSTRSTAT.MPROG</code> indicates whether or not a master transfer is in progress. If clear (<code>TWI_MSTRSTAT.MPROG = 0</code>), currently no transfer is taking place. This can occur after a transfer is complete or while an enabled master is waiting for an idle bus.
		0 No Status
		1 Master Transfer in Progress

Rx Data Double-Byte Register

The `TWI_RXDATA16` holds a 16-bit data value read from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

The data is read in little endian byte order, where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (`TWI_FIFOSTAT.RXSTAT`) field is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the read data is unknown and the existing FIFO buffer data and its status remains unchanged.

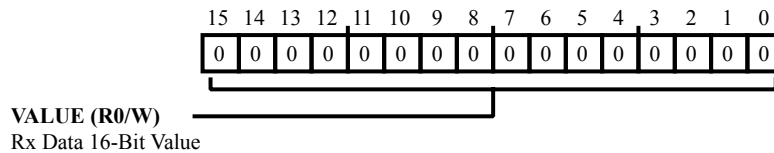


Figure 21-23: TWI_RXDATA16 Register Diagram

Table 21-16: TWI_RXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Rx Data 16-Bit Value.

Rx Data Single-Byte Register

The `TWI_RXDATA8` holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in first-out order. Although peripheral bus reads are 16 bits, a read access to `TWI_RXDATA8` accesses only one transmit data byte from the FIFO buffer. With each access, the receive status (`TWI_FIFOSTAT.RXSTAT`) field is updated. If an access is performed while the FIFO buffer is empty, the data is unknown and the FIFO buffer status remains indicating it is empty.

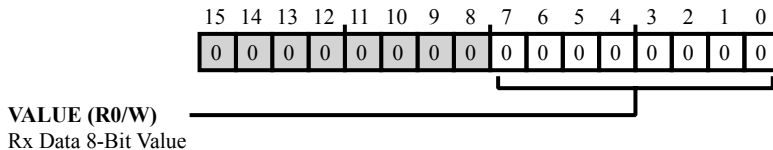


Figure 21-24: TWI_RXDATA8 Register Diagram

Table 21-17: TWI_RXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Rx Data 8-Bit Value.

Slave Mode Address Register

The `TWI_SLVADDR` holds the slave mode address, which is the valid address to which the slave-enabled TWI controller responds. The TWI controller compares this value with the received address during the addressing phase of a transfer.

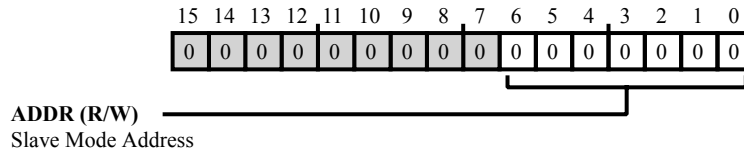


Figure 21-25: TWI_SLVADDR Register Diagram

Table 21-18: TWI_SLVADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Slave Mode Address.

Slave Mode Control Register

The `TWI_SLVCTL` controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

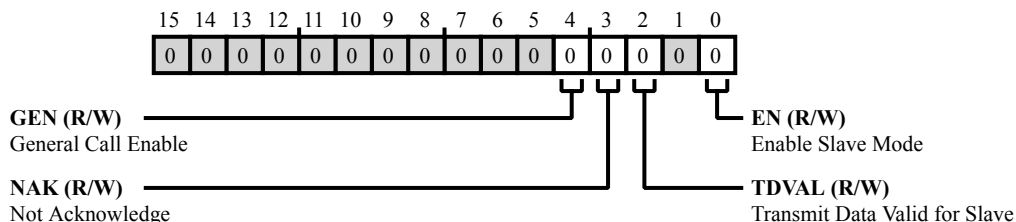


Figure 21-26: TWI_SLVCTL Register Diagram

Table 21-19: TWI_SLVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	GEN	General Call Enable. The <code>TWI_SLVCTL.GEN</code> enables general call address matching. When enabled, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated. Note that general call address detection is available only when slave mode is enabled.
		0 Disable General Call Matching
		1 Enable General Call Matching
3 (R/W)	NAK	Not Acknowledge. The <code>TWI_SLVCTL.NAK</code> directs the TWI to generate a NAK (if set) or an ACK (if cleared) at the conclusion of data transfer for slave receive. For NAK, the slave is still considered to be addressed at the conclusion of transfer.
		0 Generate ACK
		1 Generate NAK
2 (R/W)	TDVAL	Transmit Data Valid for Slave. The <code>TWI_SLVCTL.TDVAL</code> selects whether the data in the transmit FIFO is available (valid) for slave transmission (<code>TWI_SLVCTL.TDVAL</code> set). If the FIFO data is not available (invalid) for slave transmission (<code>TWI_SLVCTL.TDVAL</code> cleared), the data in the transmit FIFO is for master mode transmits, and the data is not allowed to be used during a slave transmit; the transmit FIFO is treated as if it is empty.
		0 Data Invalid for Slave Tx
		1 Data Valid for Slave Tx

Table 21-19: TWI_SLVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Slave Mode. The <code>TWI_SLVCTL.EN</code> enables slave operation. Enabling slave and master modes of operation concurrently is allowed. If disabled, no attempt is made to identify a valid address. If <code>TWI_SLVCTL.EN</code> is cleared during a valid transfer, clock stretching ceases, the serial data line is released, and the current byte is not acknowledged.	
		0	Disable
		1	Enable

Slave Mode Status Register

During and at the conclusion of register slave mode transfers, the `TWI_SLVSTAT` holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupt requests. Master mode operation does not affect slave mode status bits.

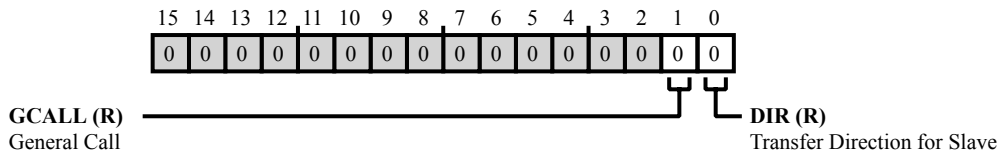


Figure 21-27: TWI_SLVSTAT Register Diagram

Table 21-20: TWI_SLVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	GCALL	General Call. The <code>TWI_SLVSTAT.GCALL</code> indicates whether or not--at the time of addressing--the address was determined to be a general call. This bit self clears if slave mode is disabled (<code>TWI_SLVCTL.EN = 0</code>).
		0 Not a General Call Address
		1 General Call Address
0 (R/NW)	DIR	Transfer Direction for Slave. The <code>TWI_SLVSTAT.DIR</code> indicates whether--at the time of addressing--the transfer direction was determined to be slave transmit or receive. This bit self clears if slave mode is disabled (<code>TWI_SLVCTL.EN = 0</code>).
		0 Slave Receive
		1 Slave Transmit

Tx Data Double-Byte Register

The `TWI_TXDATA16` register holds a 16-bit data value written into the FIFO buffer. To reduce interrupt latency output rates and peripheral bus access times, a double byte transfer data access can be done. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little endian byte order, where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (`TWI_FIFOSTAT.TXSTAT`) field is updated. If an access is performed while the FIFO buffer is not empty, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register when read back returns zero.

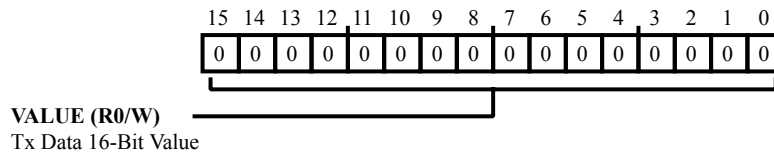


Figure 21-28: TWI_TXDATA16 Register Diagram

Table 21-21: TWI_TXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Tx Data 16-Bit Value.

Tx Data Single-Byte Register

The `TWI_TXDATA8` register holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in first-out order. For 16-bit peripheral bus writes, a write access to this register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (`TWI_FIFOSTAT.TXSTAT`) field is updated. If an access is performed while the FIFO buffer is full, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register returns zero when read back.

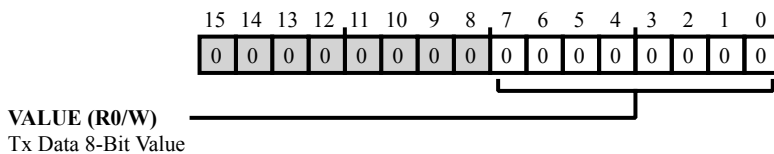


Figure 21-29: TWI_TXDATA8 Register Diagram

Table 21-22: TWI_TXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Tx Data 8-Bit Value.

22 Digital Audio Interface (DAI)

The Digital Audio Interfaces (DAI_n) are comprised of groups of identical peripherals and their respective Signal Routing Units (SRU_n). The SRU connects inputs and outputs of the DAI peripherals with each other and to the external pins. This configuration allows peripherals to be interconnected to accommodate a wide variety of systems without making external pin connections.

In a typical processor, static (multiplexed) pins are assigned to specific peripherals. When certain peripherals are not required for an application, these pins are unnecessary and expensive. The pins may need to be defined as high or low to prevent any illegal conditions. The signal routing units on the SHARC processors address this situation by controlling a number of general-purpose pins which can be assigned flexibly (a virtual connectivity between peripherals) depending on system requirements. This virtual connectivity includes pin buffers and routing logic (multiplexer). It also allows the SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

SRU Features

The SRU has the following features and capabilities.

- Flexible connections that can be made through software and during run time; no hard-wiring is required.
- At reset, a default routing scheme is already programmed.
- Connectivity can be made internally between peripherals, externally between pin buffers, or a mix of both.
- Status of the pin buffers can be programmed for conditional execution or interrupts.
- Some pin buffers allow control of signal polarity changes.
- No fan-out limitation, a peripheral or pin buffer output that can be routed to multiple peripheral or pin buffer inputs.
- Two independent routing systems are available: the DAI0 and the DAI1. Signals cannot be interconnected between both routing units with the exception of the following modules under some specific scenarios.
 - The precision clock generator (PCG)
 - A pair of DAI pin buffers from each DAI unit
 - Asynchronous Sample Rate Converter (ASRC) data for daisy chaining across the DAI units

NOTE: The 400-ball CSP BGA package has 2 x 14 DAI pins bonded off-chip (DAIx_PIN01 through DAIx_PIN12, DAIx_PIN19, and DAIx_PIN20). The 120-lead LQFP-EP package has 2 x 12 DAI pins bonded off-chip (DAIx_PIN01 through DAIx_PIN10, DAIx_PIN19, and DAIx_PIN20).

Functional Description

The fundamental timing clock of the DAI modules is SCLK.

The [DAI Block Diagram](#) shows how the DAI pin buffers are connected through the SRUn. This configuration allows for flexible signal routing.

The DAI units are comprised of four primary blocks:

- Peripherals (A/B/C) associated with DAI_n
- Signal Routing Units (SRUn)
- DAI_n I/O pin buffers
- Miscellaneous buffers

The peripherals shown in [DAI Block Diagram](#) can have up to three connections (if master or slave capable); one acts as a signal input, one as a signal output and the third as an output enable. The SRUs are based on a group of multiplexers which are controlled by registers to establish the desired interconnections. The DAI pin buffers have three signals which are used for input and output to or from off-chip and the third for output enable.

The miscellaneous buffers have an input and an output and are used for group interconnection.

The figures are a simplified representation of a DAI system. In a real representation, the SRU and DAI would show several types of data being routed from several sources including the following:

- Serial ports (SPORT)
- Precision clock generators (PCG)
- Asynchronous sample rate converters (SRC)
- S/PDIF transmitter
- S/PDIF receiver
- DAI interrupts (miscellaneous)
- PDM interface

ADSP-2156x DAI Register List

The Digital Audio Interfaces (DAI_n) contain groups of identical peripherals which can be connected internally between peripherals, externally between pin buffers, or a mix of both. This module contains the following registers.

Table 22-1: ADSP-2156x DAI Register List

Name	Description
DAI_CLK0	Clock Routing Control Register 0
DAI_CLK1	Clock Routing Control Register 1
DAI_CLK2	Clock Routing Control Register 2
DAI_CLK3	Clock Routing Control Register 3
DAI_CLK4	Clock Routing Control Register 4
DAI_CLK5	Clock Routing Control Register 5
DAI_DAT0	Serial Data Routing Control Register 0
DAI_DAT1	Serial Data Routing Control Register 1
DAI_DAT2	Serial Data Routing Control Register 2
DAI_DAT3	Serial Data Routing Control Register 3
DAI_DAT4	Serial Data Routing Control Register 4
DAI_DAT5	Serial Data Routing Control Register 5
DAI_DAT6	Serial Data Routing Control Register 6
DAI_FS0	Frame Sync Routing Control Register 0
DAI_FS1	Frame Sync Routing Control Register 1
DAI_FS2	Frame Sync Routing Control Register 2
DAI_FS4	Frame Sync Routing Control Register 4
DAI_GBL_INT_EN	Global SPORT Interrupt Grouping Register
DAI_GBL_SP_EN	Global SPORT Enable Register
DAI_IMSK_FE	Falling-Edge Interrupt Mask Register
DAI_IMSK_PRI	Core Interrupt Priority Assignment Register
DAI_IMSK_RE	Rising-Edge Interrupt Mask Register
DAI_IRPTL_H	High Priority Interrupt Latch Register
DAI_IRPTL_HS	Shadow High Priority Interrupt Latch Register
DAI_IRPTL_L	Low Priority Interrupt Latch Register
DAI_IRPTL_LS	Shadow Low Priority Interrupt Latch Register
DAI_MISCO	Miscellaneous Control Register 0
DAI_MISC1	Miscellaneous Control Register 1
DAI_PBEN0	Pin Buffer Enable Register 0
DAI_PBEN1	Pin Buffer Enable Register 1
DAI_PBEN2	Pin Buffer Enable Register 2

Table 22-1: ADSP-2156x DAI Register List (Continued)

Name	Description
DAI_PBEN3	Pin Buffer Enable Register 3
DAI_PIN0	Pin Buffer Assignment Register 0
DAI_PIN1	Pin Buffer Assignment Register 1
DAI_PIN2	Pin Buffer Assignment Register 2
DAI_PIN4	Pin Buffer Assignment Register 4
DAI_PIN_STAT	Pin Status Register

ADSP-2156x DAI Interrupt List

Table 22-2: ADSP-2156x DAI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
20	DAI0_IRQH	DAI0 High priority	None	
21	DAI1_IRQH	DAI1 High priority	None	
116	DAI0_IRQL	DAI0 Low Priority Interrupt	None	
117	DAI1_IRQL	DAI1 Low Priority Interrupt	None	
189	DAI0_GBL_SPORT_INT0	DAI0 DAI0 Global SPORT interrupt0	None	
190	DAI0_GBL_SPORT_INT1	DAI0 DAI0 Global SPORT interrupt1	None	
191	DAI1_GBL_SPORT_INT0	DAI1 DAI1 Global SPORT interrupt0	None	
192	DAI1_GBL_SPORT_INT1	DAI1 DAI1 Global SPORT interrupt1	None	

ADSP-2156x DAI Trigger List

Table 22-3: ADSP-2156x DAI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
105	DAI0_GBL_SPORT_TRG_00	DAI0 SPORT GROUP0 Trigger Output	None
106	DAI0_GBL_SPORT_TRG_01	DAI0 SPORT GROUP1 Trigger Output	None
107	DAI1_GBL_SPORT_TRG_00	DAI1 SPORT GROUP2 Trigger Output	None
108	DAI1_GBL_SPORT_TRG_01	DAI1 SPORT Group3 Trigger Output	None

Table 22-4: ADSP-2156x DAI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

DAI Block Diagram

The *DAI Functional Block Diagram* and the *Digital Audio Interconnect Unit* figures show the functional blocks within the DAI and the unit connections to the peripherals.

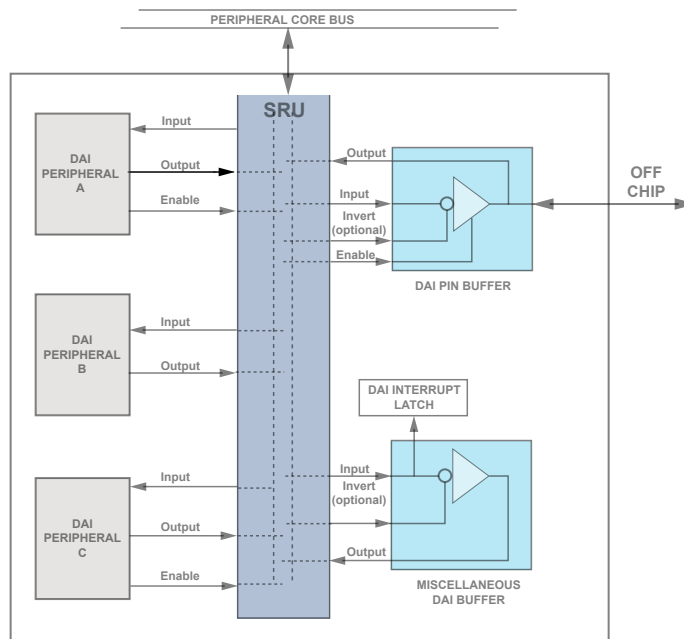


Figure 22-1: DAI Functional Block Diagram

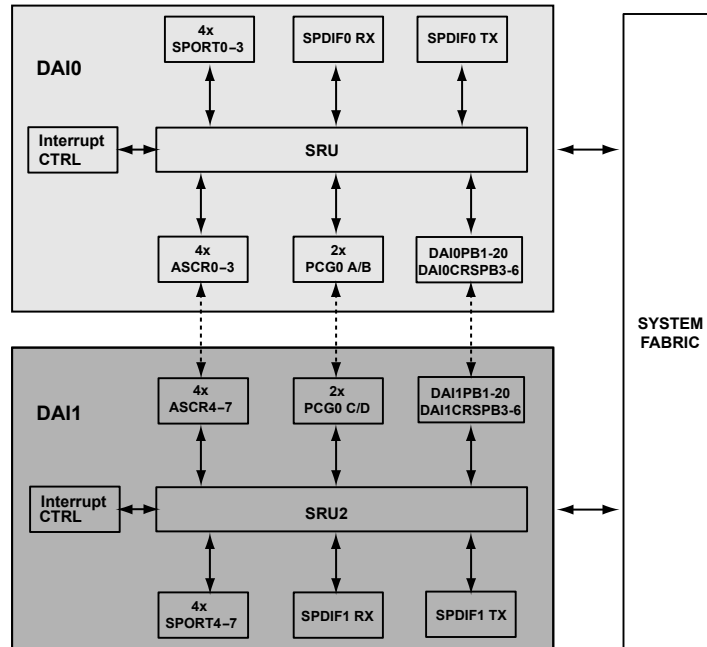


Figure 22-2: Digital Audio Interconnect Unit

DAI Signal Naming Conventions

The peripherals associated with the DAI do not have any dedicated I/O pins for off-chip communication. Instead, the I/O pin is only accessible in the chip internally and is known as an *internal node*. Every internal node of a DAI peripheral (input or output) is given a unique mnemonic. The convention is to begin the name with an identifier for the peripheral that the signal is coming to or from, followed by the function of the signal.

A number is included if the DAI contains more than one peripheral type (for example, serial ports), or if the peripheral has more than one signal that performs this function (for example, serial ports). The mnemonic always ends with *_I* if the signal is an input, or with *_O* if the signal is an output. An example is shown in the *Example DAI Mnemonics* figure.

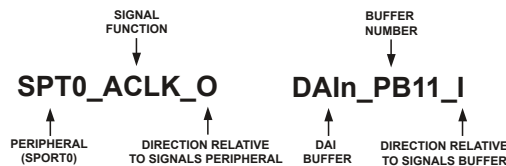


Figure 22-3: Example DAI Mnemonics

I/O Pin Buffers

Within the context of the SRU, physical connections to the DAI pins are replaced by a logical interface known as a *pin buffer*. This three terminal active device is capable of sourcing or sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has an input, an output, and an enable as shown in the *Pin Buffer Example* figure. The inputs and the outputs are defined with respect to the pin, similar to a peripheral device. This naming convention is consistent with the SRU naming convention.

Pin Buffer Signals

The pin buffer is based on three signals shown in the *Pin Buffer Example* figure and described in the following sections.

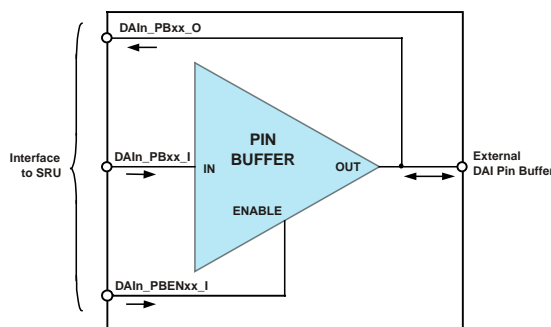


Figure 22-4: Pin Buffer Example

Pin Buffer Input Signal

A pin buffer input ($DAIn_PBxx_I$) is driven as an output from the processor when the pin buffer enable is set ($=1$). Each physical pin (connected to a bonded pad) can be connected through the SRU to any of the outputs of the DAI peripherals, based on the bit field values. The SRU can also be used to route signals that control the pins in other ways. Many signals can be configured for use as control signals.

Pin Buffer Enable Signal

When a pin buffer enable ($DAIn_PBENxx_I$) is set ($=1$), the signal present at the corresponding pin buffer input ($DAIn_PBxx_I$) is driven off-chip as an output. When a pin buffer enable is cleared ($=0$), the signal present at the corresponding pin buffer input is ignored. The pin enable control registers activate the drive buffer for each of the DAI pins. When the pins are not enabled (driven), they can be used as inputs. There are two options to control the pin buffer enable signal; setting the level high for a static solution, or connecting the dedicated peripheral's pin buffer output enable signal to its pin buffer, which automatically enables the pin buffer.

Pin Buffer Functions

Pin buffers can be configured as inputs or outputs as described in the following sections.

Pin Buffers as Signal Input

When the DAI pin is used only as an input, connect the corresponding pin buffer enable to logic low as shown in the *Pin Buffer as Input* figure. This configuration disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI pin and at the pin buffer output. When the pin buffer enable (for example, $DAI_PBEN0.PB01$) is cleared ($=0$), the pin buffer output ($DAIn_PBxx_O$) is the signal driven onto the DAI pin by an external source, and the pin buffer input signal ($DAIn_PBxx_I$) is not used.

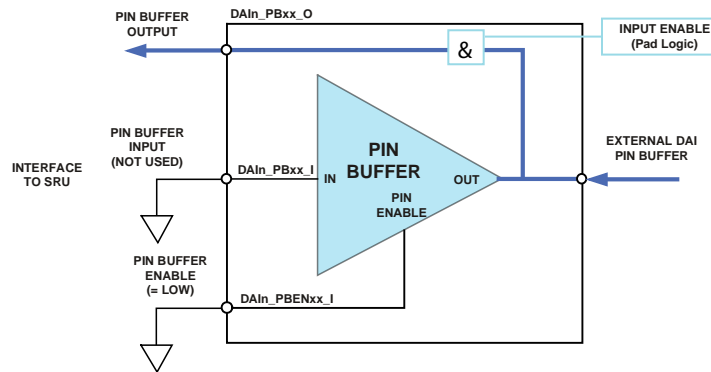


Figure 22-5: Pin Buffer as Input

NOTE: Whether programmed as input or output, a DAI buffer input always routes the same signal to an output internally. DAI pins have programmable internal pull-up resistors. For more information, see the General-Purpose Ports (PORT) chapter for details.

Pin Buffers As Signal Output

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they can be used in either direction. Each of the DAI pins can be used as either an output or an input. Although the direction of a DAI pin is set simply by writing to a memory-mapped register, most often the direction of the pin is dictated by the designated use of that pin.

When the DAI pin is used only as an output, connect the corresponding pin buffer enable to logic high as shown in the *Pin Buffer as Output* figure. This configuration enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer input onto the DAI pin and off-chip. When the pin buffer enable bits are set (in the `DAIn_PBxx_I` registers) ($=1$), the pin buffer output (`DAIn_PBxx_O`) is the same signal as the pin buffer input (`DAIn_PBxx_I`), and this signal is driven as an output.

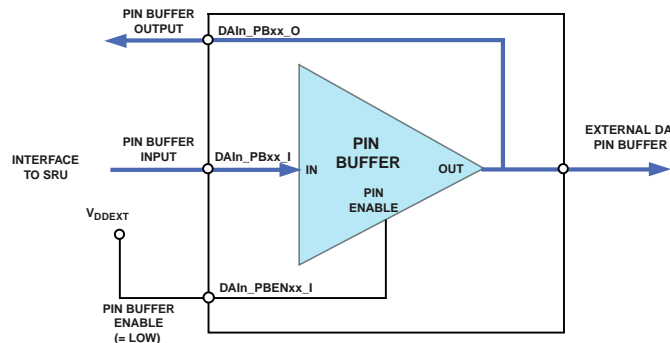


Figure 22-6: Pin Buffer as Output

DAI Pin Buffer Status

The signal levels on the DAI pins can be read with the `DAI_PIN_STAT` registers.

DAIn Peripherals

There are two categories of peripherals associated with the DAI units. These are described in the following sections.

Output Signals With Pin Buffer Enable Control

Many peripherals within the DAI that have bidirectional pins generate a corresponding pin enable signal. Typically, the settings within the control registers of a peripheral determine if a bidirectional pin is an input or an output. The pin is then driven accordingly.

Though most peripherals are capable of operating bidirectionally, it is not required that all of the `_I` and `_O` signals of a peripheral be connected to the pin buffer. If the system design only uses a signal in one direction, it is simpler to connect the pin buffer accordingly.

NOTE: All available pin buffer output enables must be routed to their pin buffer input enable signals in cases where data streaming connections are used. This arrangement guarantees timing requirements.

NOTE: In some cases, it is necessary to use a peripheral's dedicated pin buffer enable signal instead of static levels. For example, SPORT TDM mode requires the SPORT's dedicated data pin buffer enable signal to be used for the SPORT's data pin to three-state the data pins on inactive channels.

Output Signals Without Pin Buffer Enable Control

Some peripherals have signal outputs without an automated pin buffer control enable signal. The operation of these peripherals is simplified. The routing to a DAI pin buffer enable input requires a static high from the SRUn. In order to disable the pin buffer output, software must clear the pin buffer enable input accordingly.

Signal Routing Units (SRUs)

The following sections provide details specific to the SRUs.

Cross Mode Connections

The DAI allows a few signals to be interconnected across DAI units. These connections are commonly referred as *cross mode connections*. These cross mode connections are useful for the system designer to share signals across both DAIs so that synchronization can be achieved on the peripheral modules across DAI0 and DAI1.

NOTE: The 400-ball CSP BGA package has 2 x 14 DAI pins bonded off-chip (DAIx_PIN01 through DAIx_PIN12, DAIx_PIN19, and DAIx_PIN20). The 120-lead LQFP-EP package has 2 x 12 DAI pins bonded off-chip (DAIx_PIN01 through DAIx_PIN10, DAIx_PIN19, and DAIx_PIN20).

Shared Clock

This cross mode connection (see [Figure 22-7 Cross Connect Clock Example](#)) can be used to share a clock signal across DAIs. DAI0_PIN03/DAI0_PIN05 and DAI1_PIN03/DAI1_PIN05 can be used to share clock signal across DAI. When DAI0_PIN03/DAI0_PIN05 is configured as input clock pin then this clock signal is available in Group A of both DAIs. This source signal is DAI0_PB03_O/DAI0_PB05_O in group A of DAI 0 and DAI1_CRS_PB03_O/DAI1_CRS_PB05_O in group A of DAI1.

Peripherals routed through DAI0 like PCG A and SPORT 0A can use DAI0_PB03_O/DAI0_PB05_O as the input clock signal. Peripherals routed through DAI1 like PCG C, SPORT 4A can use DAI1_CRS_PB03/DAI1_CRS_PB05 as the input clock signal. Similarly when DAI1_PIN03 is configured as the input clock pin, then this clock signal is available as DAI1_PB03_O/DAI1_PB05_O in group A of DAI1 and DAI0_CRS_PB03/DAI0_CRS_PB05 in group A of DAI0.

Shared Frame Sync

This cross mode connection can be used to share a frame sync signal cross DAIs. DAI0_PIN04/DAI0_PIN06 and DAI1_PIN04/DAI1_PIN06 can be used to share the frame sync signal across the DAIs. When DAI0_PIN04/DAI0_PIN06 is configured as the input frame sync pin, then this frame sync signal is available in Group C of both DAIs. This source signal is DAI0_PB04_O/DAI0_PB06_O in group C of DAI 0 and DAI1_CRS_PB04_O/DAI1_CRS_PB06_O in group C of DAI1.

Peripherals on DAI0 like SPORT 0A can use DAI0_PB04_O/DAI0_PB06_O as the input frame sync signal. The peripherals on DAI1 like SPORT 4A can use DAI1_CRS_PB04/DAI1_CRS_PB06 as the input sync signal. Similarly, when DAI1_PIN04 /DAI1_PIN06 is configured as input frame sync pin, then this frame sync signal is available as DAI1_PB04_O/DAI1_PB06_O in group A of DAI1 and DAI0_CRS_PB04/DAI0_CRS_PB06 in group C of DAI0.

Shared Pin Buffer

This cross mode connection can be used to connect input signals on DAI0_PIN03/DAI0_PIN05, DAI0_PIN04/DAI0_PIN06, DAI1_PIN03/DAI1_PIN05 and DAI1_PIN04 /DAI1_PIN06 to other DAI pins across DAI. When DAI0_PIN03/DAI0_PIN05 and DAI0_PIN04 are configured as input signals, these signals can be routed to the pin buffers of DAI1 in Group D using DAI1_CRS_PIN03/DAI1_CRS_PIN05 and DAI1_CRS_PIN04. When DAI1_PIN03/DAI1_PIN05 and DAI1_PIN04/DAI1_PIN06 are configured as input signals, they can be routed to the pin buffers of DAI0 in Group D using DAI0_CRS_PIN03/DAI0_CRS_PIN05 and DAI0_CRS_PIN04/DAI0_CRS_PIN06 signals.

PCGs and Pin Buffers

This cross mode connection (see [Figure 22-8 Cross Connect PCG Example](#)) can be used to connect PCG output signals of one DAI to pin buffers of other DAI. For example, PCG C outputs of DAI 1 can be connected to pin buffers of DAI 0 using PCG0_CRS_CLKC_O of Group D.

ASRC TDM and Data Signal

This cross mode connection is available in Group B (Data Signals). This allows programs to connect the Data and TDM signals of SRC3 and SRC 7 across DAIs so that eight SRCs can be daisy chained for TDM 16 operation. For example SRC3_DAT_OP_O and SRC3_TDM_IP_O of DAI0 are available as SRC3_CRS_DAT_OP_O and SRC3_CRS_IP_O in DAI1 so that DAI0 SRC3 can be connected to DAI1's SRC for daisy chaining.

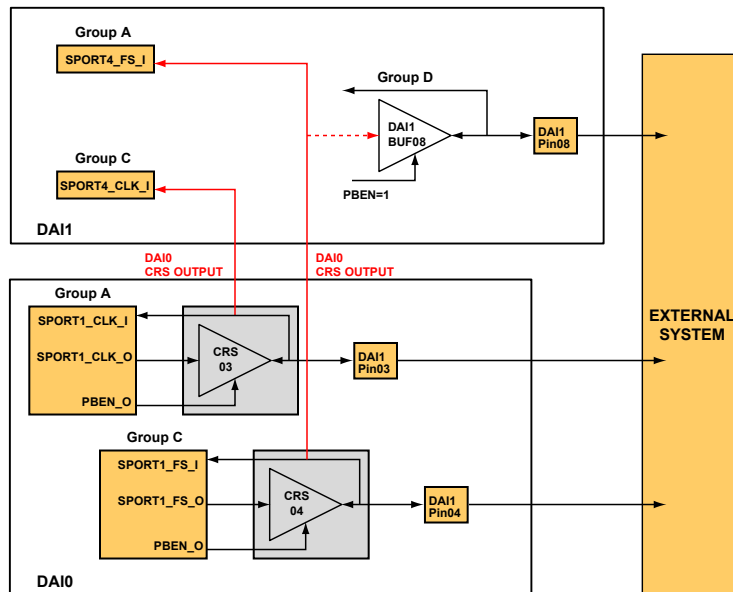


Figure 22-7: Cross Connect Clock Example

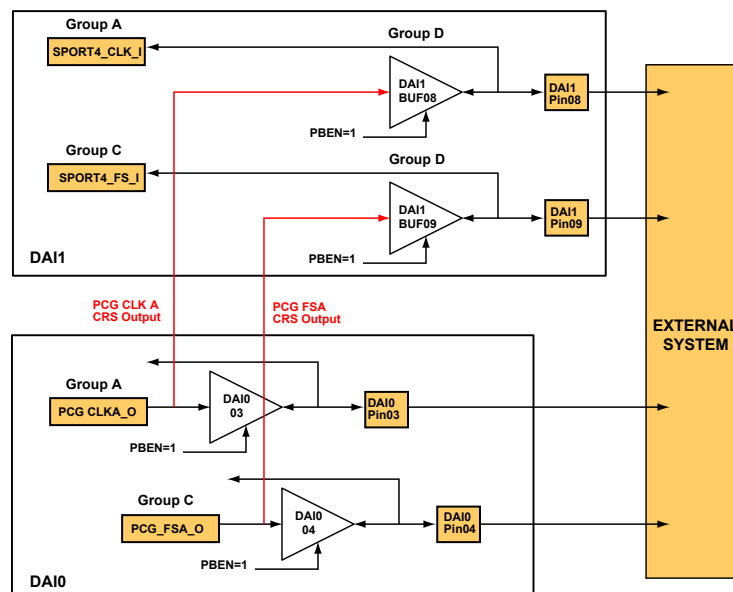


Figure 22-8: Cross Connect PCG Example

Refer to the [DAI Group Routing](#) section for more details on DAI routing.

Signal Routing Matrix by Groups

The SRU is similar to a set of patch bays, which contain a bank of inputs and a bank of outputs. For each input (destination), there is a set of permissible output (source) options. Outputs can feed to any number of inputs in parallel, but every input must be patched to exactly one valid output source. Together, the set of inputs and outputs are called a group. The signal's inputs and outputs that comprise each group all serve similar purposes. They are

compatible such that almost any output-to-input patch makes functional sense. With the grouping, the multiplexing scheme becomes highly efficient since it does not make sense (for example) to route a frame sync signal to a data signal.

The SRU for the DAI contains six groups named A through F. Each group routes a unique set of signals with a specific purpose:

- Group A routes clock signals
- Group B routes serial data signals
- Group C routes frame sync signals
- Group D routes pin signals
- Group E routes miscellaneous signals
- Group F routes pin output enable signals

Together, the six groups of the SRU include all of the inputs and outputs of the DAI peripherals, a number of additional signals from the core, and all of the connections to the DAI pins.

NOTE: It is not possible to connect a signal in one group directly to a signal in a different group (analogous to wiring from one patch bay to another). However, group D (DAI) is largely devoted to routing in this vein.

DAI Group Routing

Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, DAI group A is used to route clock signals. The memory-mapped group A registers, `DAI_CLK0` through `DAI_CLK5`, contain bit fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems). The *Example DAI Group A Multiplexing (DAI_CLKx)* diagrams the input signals that are controlled by the group A registers. All bit fields in the SRU configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the SRU.

The SRU is similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRU, a bit pattern that is associated with a signal output (shown in the *Example DAI Group A Multiplexing (DAI_CLKx)* figure) is written to a bit field corresponding to a signal input.

The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

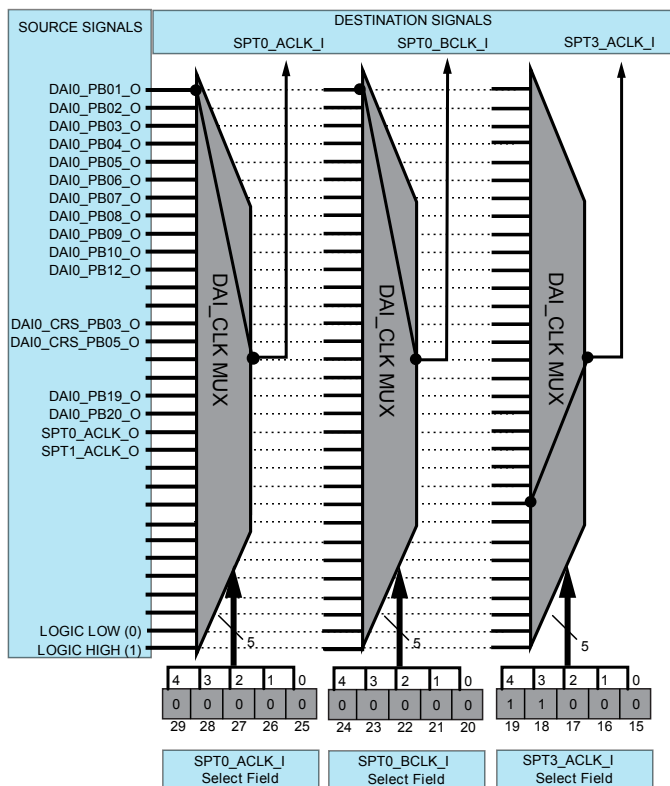


Figure 22-9: Example DAI Group A Multiplexing (DAI_CLKx)

Just as group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. All of the groups have an encoding that allow a signal to flow from a pin output to the input being specified by the bit field.

Group D routes signals to pins so that they may be driven off-chip (required to route a signal to the pin input). Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. The input of one pin can be patched to the output of another pin, allowing board-level routing under software control.

Rules for SRU Connections

There are two rules which apply to all routing:

1. One source (output node) can drive different destinations (input nodes).
2. One destination (input node) can only be assigned to one source (output node).

As an example from the *Example DAI Group A Multiplexing (DAI_CLKx)* figure:

- DAI0_PB02_O is routed to SPT0_ACLK_I
- DAI0_PIN02_O is routed to SPT0_BCLK_I
- SPT2_ACLK_O is routed to SPT3_ACLK_I

NOTE: Inputs may only be connected to outputs.

Miscellaneous Buffers and Functions

The SRU group E provides miscellaneous buffers used for group interconnect.

DAI group E connections are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals and provides a means of connecting signals between groups.

In the *DAI MISCAx SRU Signal Connections* table, the `DAIn_MISCAx_I` signals appear as inputs in group E (also connected to the DAI interrupt logic), but do not directly feed any peripheral. Rather, the `MISCAx_O` signals reappear as outputs in group F.

Table 22-5: DAI MISCAx SRU Signal Connections

MISCA Source	DAI Connection	MISCA Destination
	Group E	DAIn_MISCA5-0_I
DAIn_MISCA5-0_O	Group F	

Additional connections among groups provide a great amount of utility. Since the output groups F (DAI) dictate pin direction, these few signal paths enable a number of possible uses and connections for the DAI pins. Other examples include:

- A pin input can be patched to another pin's enable, allowing an off-chip signal to gate an output from the processor.
- Any of the DAI pins can be used as interrupt sources or general-purpose I/O (GPIO) signals.

In summary, the SRU enables many possible functional changes, both within the processor as well as externally. Used creatively, it allows system designers to radically change functionality at run time, and to potentially reuse circuit boards across many products.

DAI Routing Capabilities

This section describes the routing options to aid in designing a system using the DAI units. The [DAI Default Routing](#) section provides diagrams that show how that DAIs connect at default. The DAI Group tables provide the source signals and selections codes. Finally, the provides information about configuring destinations.

The *DAI Routing Capabilities* tables provide an overview of the different routing capabilities for the DAI unit. For information on an individual peripherals routing, see the "SRU Programming" section of the specific peripheral chapter.

Table 22-6: DAI0 Routing Capabilities

Source Signals - Output (xxxx_O)		DAI0 Group	Destination Signals - Input (xxxx_I)
SPORT2B-0A (clocks) PCG A, B S/PDIF0 Rx (clock, TDM clock)	DAI0 pin buffer 12-1 and 20-19 DAI0 CRS pin buffer (buffer 3 and 5 from other DAI) Logic level high Logic level low	A-clocks	SPORT3B-0A (clocks) SRC3-0 (clocks) PCG A-B ext. clock, ext. sync S/PDIF0 Tx (clock, HF clock)
SPORT3B-0A (data) SRC3-0 (data, TDM data) S/PDIF0 Tx/Rx (data) SRC7 from other DAI (data, TDM data)	DAI0 pin buffer 12-1 and 20-19 Logic level high Logic level low	B-data	SPORT3B-0A (data) SRC3-0 (data, TDM data) S/PDIF0 Tx/Rx (data)
SPORT2B-0A (FS) PCG A, B (FS) S/PDIF0 Rx (FS)	DAI0 pin buffer 12-1 and 20-19 DAI0 CRS pin buffer (buffer 4 and 6 from other DAI) Logic level high Logic level low	C-frame sync	SPORT3B-0A (FS) SRC3-0 (FS)
SPORT3B-0A (clock, FS, TDV, data) PCG A,B (clock, FS) S/PDIF0 Rx (clock, TDM clock, FS, data) S/PDIF0 Tx (data, block start) PCG C, D from the other DAI (clock, FS)	DAI0 pin buffer 12-1 and 20-19 DAI0 pin buffer 3, 4, 5, and 6 from other DAI Logic level high Logic level low	D-pin buffer inputs	DAI0 pin buffer 12-1 and 20-19 Options: DAI0 pin buffer 20-19 Polarity change
SPORT2B-0A (FS) PCG A (clock) PCG B (clock, FS) S/PDIF0 Tx (block start)	DAI0 pin buffer 12-1 and 20-19 Logic level high Logic level low	E-miscellaneous signals	<i>DAI Interrupt 31-22</i> MISCA5-0 Options: MISCA5- Polarity change
SPORT3B-0A (clock, FS, data, TDV) MISCA5-0	DAI0 pin buffer 12-1 and 20-19 Logic level high Logic level low	F-pin buffer enable	DAI0 pin buffer enable 12-1 and 20-19

Table 22-7: DAI1 Routing Capabilities

Source Signals - Output (xxxx_O)		DAI1 Group	Destination Signals - Input (xxxx_I)
SPORT6B–4A (clocks) PCG C, D (clocks) S/PDIF1 Rx (clock, TDM clock)	DAI1 Pin Buffer 12-1 and 20–19 DAI1 CRS pin buffer (buffer 3 from other DAI) Logic level high Logic level low	A-Clocks	SPORT7B–4A (clocks) SRC4–7 (clocks) PCG C–D Ext. clock, Ext. sync S/PDIF1 Tx (clock, HF clock)
SPORT6B–4A (data) SRC4–7 (data, TDM data) S/PDIF1 Tx/Rx (data) SRC3 from other DAI (data, TDM data)	DAI1 Pin Buffer 12-1 and 20–19 Logic level high Logic level low	B-Data	SPORT7B–4A (data) SRC4–7 (data, TDM data) S/PDIF1 Tx/Rx (data)
SPORT6B–4A (FS) PCG C, D (FS) S/PDIF1 Rx (FS)	DAI1 Pin Buffer 12-1 and 20–19 DAI1 CRS pin buffer (buffer 4 and 6 from other DAI) Logic level high Logic level low	C-Frame Sync	SPORT7B–4A (FS) SRC4–7 (FS)
SPORT6B–4A (clock, FS, TDV, data) PCG C, D (clock, FS) S/PDIF1 Rx (clock, TDM clock, FS, data) S/PDIF1 Tx (data, block start) PCG A, B from the other DAI (clock, FS)	DAI1 Pin Buffer 12-1 and 20–19 DAI1 Pin Buffer 3, 4, 5, and 6 from other DAI Logic level high Logic level low	D-Pin Buffer Inputs	DAI1 Pin Buffer 12-1 and 20–19 Options: DAI1 Pin Buffer 20–19 Polarity Change
SPORT6B–4A (FS) PCG C (clock) PCG D (clock, FS) S/PDIF1 Tx (block start)	DAI1 Pin Buffer 12-1 and 20–19 Logic level high Logic level low	E-Miscellaneous Signals	<i>DAI Interrupt 31-22</i> MISCA5–0 Options: MISCA5–4 Polarity Change
SPORT7B–4A (clock, FS, data, TDV) MISCA5-0	Logic level high Logic level low	F-Pin Buffer Enable	DAI1 Pin Buffer Enable 12-1 and 20–19

DAI Default Routing

When the processor comes out of reset, the SPORT junctions are bidirectional to the DAI pin buffers. This configuration allows systems to use the SPORTs as either master or slave (without changing the routing scheme). Therefore, programs only need to use the SPORT control register settings to configure master or slave operations. Note that all DAI inputs which are not routed by default are tied to signal low.

NOTE: All DAI input buffers which are not routed by default are driven low and all DAI pin buffer enable signals are driven low.

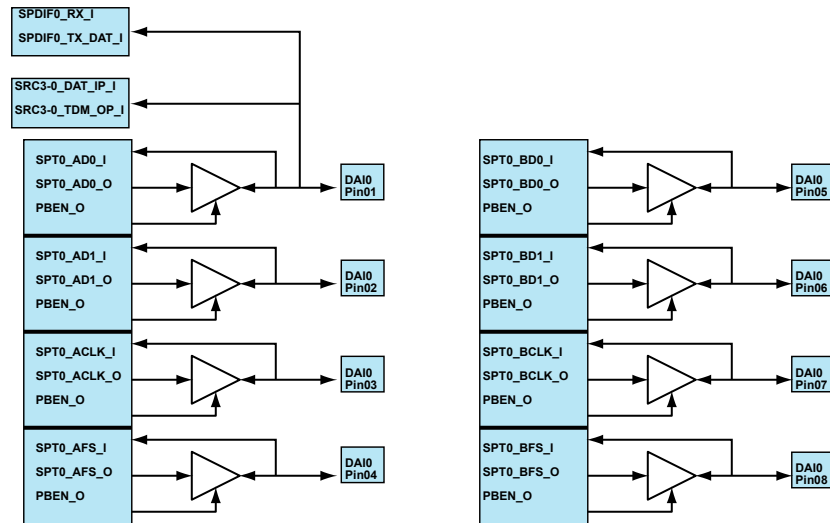


Figure 22-10: DAI0 Default Routing Pins 01-08

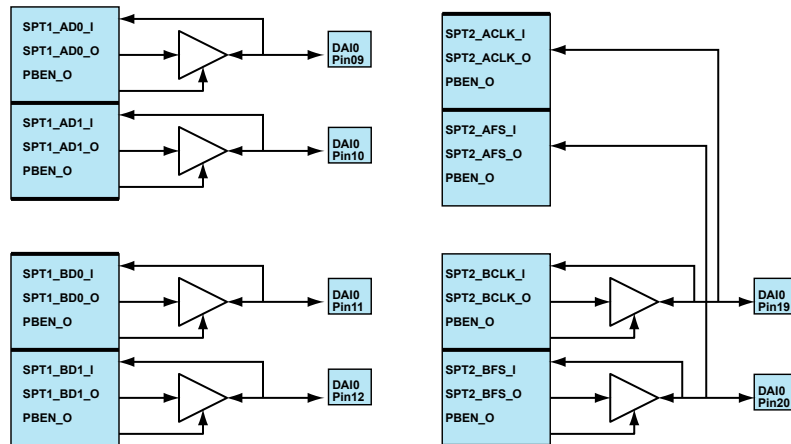


Figure 22-11: DAI0 Default Routing Pins 09-20

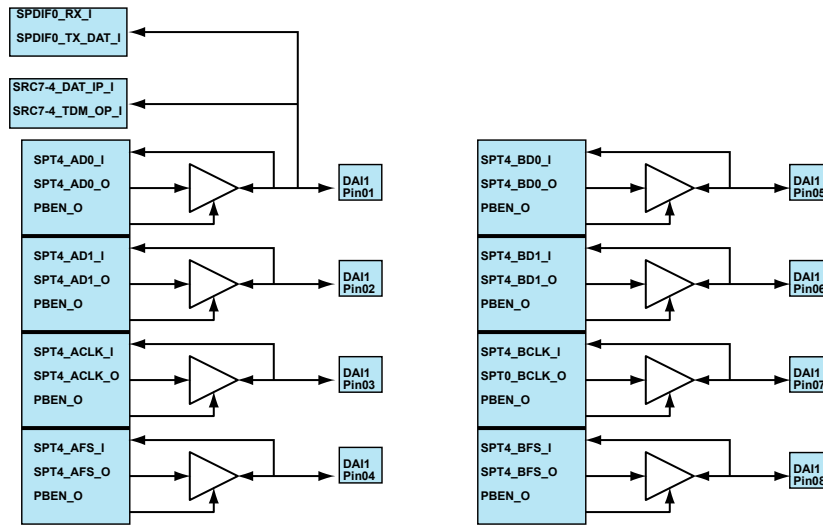


Figure 22-12: DAI1 Default Routing Pins 01-08

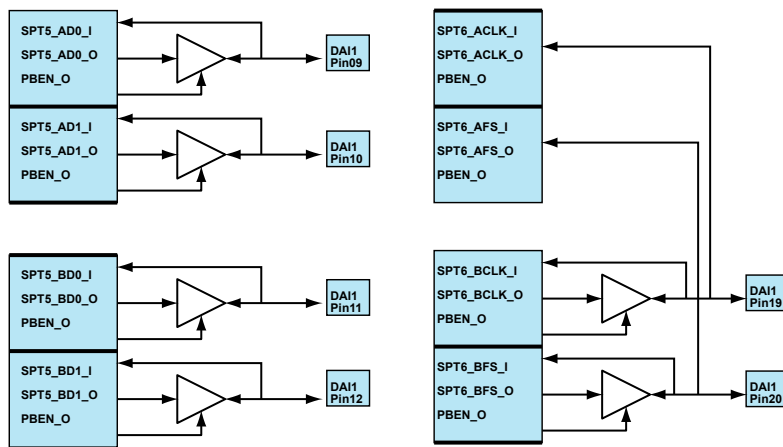


Figure 22-13: DAI1 Default Routing Pins 09-20

Unused DAI Connections

The SRUs have a default general-purpose routing scheme which can be modified to accommodate any number of different system designs. Regardless of the system design, it is good practice to tie all unused inputs to a high or low level to reduce dynamic power consumption.

DAI Operating Modes

Some buffers allow polarity changes, described as follows.

DAI Pin Buffer Polarity

As shown in the *Pin Buffer Polarity* figure, the DAI pin buffer 20-19 can change the polarity of the input signal if the corresponding control bits (DAI_PIN4 . INV20, DAI_PIN4 . INV19) are set. These bits can be set during run time and the buffer should not loopback to itself.

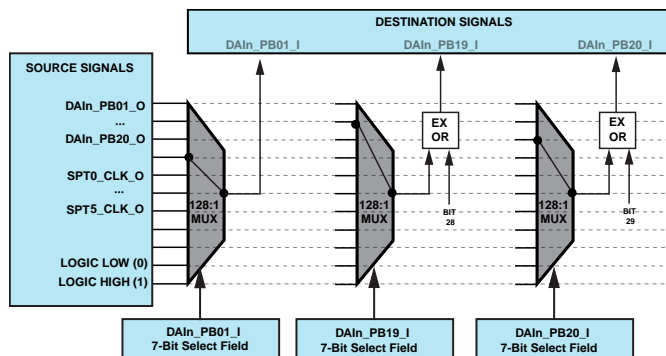


Figure 22-14: Pin Buffer Polarity

DAI Miscellaneous Buffer Polarity

As shown in the *Pin Buffer Polarity* figure, the A5-4 miscellaneous buffers can change the polarity of the input signal if the corresponding control bits (DAI_MISC1 . IN5, DAI_MISC1 . IN4) are set. Both buffers are not connected to the DAI interrupt latch register. These bits can be set during run time.

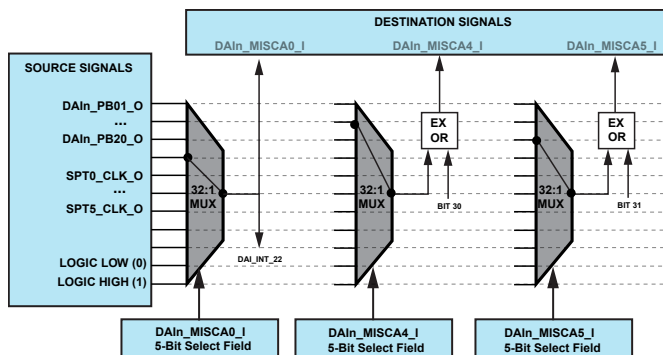


Figure 22-15: Miscellaneous Buffer Polarity

DAI System Interrupt Controller (SIC)

The DAI module incorporates a system interrupt controller (SIC) which is connected to the SEC as seen in the *DAI System Interrupt Controller* figure.

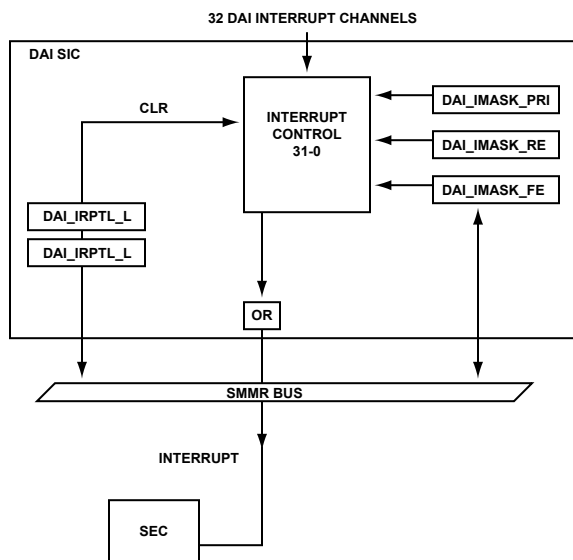


Figure 22-16: DAI System Interrupt Controller

The DAI has its own system interrupt controllers that indicate to the core when DAI audio peripheral-related events have occurred. Since audio events generally occur infrequently relative to the SHARC core, the DAI interrupt controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems. One interrupt is mapped with DAI low priority. The second interrupt is mapped with DAI high priority. This configuration allows programs to broadly indicate priority. In this way, the DAI SIC provides 32 independently configurable sources or channels. The output bus interrupt signals are logically OR'ed into one interrupt line and fed to the interrupt controller logic of the core.

Three registers are used to configure the DAI interrupt controller. Each of the 32 interrupt sources can be independently configured to trigger on a rising edge, falling edge, both edges, or neither edge of an incoming signal. All DAI interrupt control registers are memory-mapped registers and are accessed through the peripheral bus.

Interrupt Sources

The DAI's five peripheral sources are multiplexed into 32 interrupt sources and are labeled DAI_INT31-0 (*DAI Interrupt Sources* table).

NOTE: There are two naming conventions. The DAI interrupt controller register bits are labeled DAI_31-0_INT. Their corresponding SRU routing signals are labeled DAI_INT_31-0_I.

Table 22-8: DAI Interrupt Sources

Interrupt Source	Description	Signal Response
DAI_INT2-0, DAI_INT4	S/PDIF RX, 4 channels	Event
DAI_INT2-0, DAI_INT4	S/PDIF RX, 4 channels	Waveform
DAI_INT21-18	ASRC, 4 channels	
DAI_INT31-22	Miscellaneous, S/PDIF TX, 9 channels	

Interrupt Latch Priority Option

The `DAI_IMSK_PRI` register specifies the priority for the DAI interrupt channels. DAI system interrupt controller has a pair of interrupt latch registers, `DAI_IRPTL_H` and `DAI_IRPTL_L`. The configuration of the `DAI_IMSK_PRI` register also determines the interrupt latch mapping for a particular DAI interrupt. When a DAI interrupt is configured as low priority (`DAI_IMSK_PRI` bit cleared, default setting), interrupts are mapped to the `DAI_INTR_IRQL` signal and when an interrupt occurs, the corresponding bit of the `DAI_IRPTL_L` register is set. When a DAI interrupt is configured as high priority (`DAI_IMSK_PRI` bit set), interrupts are mapped to the `DAI_INTR_IRQH` signal and the interrupt is latched to the `DAI_IRPTL_H` register. The low priority DAI interrupt (`INTR_DAI_IRQL`) and high priority DAI interrupt (`INTR_DAI_IRQH`) are connected to the SEC .

Interrupt Mask for Waveforms

The `DAI_IMSK_RE` and `DAI_IMSK_FE` registers allow programs to mask or unmask interrupts for specific edges of a signal mapped to the channel. It can be configured for rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges by masking them separately. Signals from the SRU can be used to generate interrupts. For example, when the `DAI_IMSK_FE.MISCINT9` bit is set to one, any falling edge signals from the external channel generate an interrupt and the interrupt latch is set.

Interrupt Mask for Events

The system interrupt controller needs information about a peripheral's interrupt sources that correspond to event signals (see the *DAI Interrupt Sources* table). As a result, the rising edge is used as an interrupt source only. For DAI peripherals marked as events, programs may unmask an interrupt source on the rising edge only.

Shadow Interrupt Register

The DAI interrupt controller has shadow registers to simplify debug activities since these registers do not manipulate status control. Any read of the shadow registers (`DAI_IRPTL_HS`, `DAI_IRPTL_LS`), provides the same data as a read of the `DAI_IRPTL_H` and `DAI_IRPTL_L` registers. However, reads of the DAI shadow registers do not change the interrupt acknowledge status to the core interrupt controller.

Interrupt Service

The interrupt acknowledge operates differently when multiple channels are multiplexed into one interrupt output signal. When an interrupt from the DAI must be serviced, any of the two interrupt service routines (`INTR_DAI_IRQL`, `INTR_DAI_IRQH`) must query the SIC to determine the source(s). Sources can be any one or more of the DAI channels (`DAI_INT31-0`).

- When the `DAI_IRPTL_H` register is read, the high priority latched interrupts are cleared.
- When the `DAI_IRPTL_L` register is read, the low priority latched interrupts are cleared.

If an interrupt occurs in the same cycle as a latch register is cleared, the clear mechanism has lower priority and the new interrupt is registered.

Signal Routing Unit Effect Latency

After the DAI registers are configured the effect latency is 2 *SCLK0* cycles minimum and 3 *SCLK0* cycles maximum.

DAI Programming Model

As discussed in the previous sections, the signal routing unit is controlled by writing values that provide a plug-in tool in CCES so that configuring the SRU is done graphically. Analog Devices offers macros that are included with the CrossCore or VisualDSP++ tools, greatly easing code development in the SRU.

There is a macro that has been created to connect peripherals used in a DAI configuration. This code can be used in both assembly and C code. See the INCLUDE file SRU.H. In practice the macro is provided and forms the style `SRU(source_O, destination_I)` for DAI0. Example appears as: `SRU(DAI0_PB12_O, SPT0_ACLK_I);`

Debug Features

The following section describes the feature that can be used to help in debugging the DAI.

Loopback Routing

The SPORT serial peripheral supports an internal loopback mode. If the loopback bit for each peripheral is enabled, it connects the transmitter with the receiver block internally (does not signal off-chip). The SRU can be used for this purpose. The *Loopback Routing* table describes the different possible routings based on the peripheral.

NOTE: The peripheral's loopback mode for debug is independent from both of the signal routing units.

Table 22-9: Loopback Routing

Peripheral	Loopback Mode	SRU1-0 Internal Routing for Loopback	SRU1-0 External Routing for Loopback
SPORT	Yes	SPTx_xx_O → SPTx_xx_I	SPTx_xx_O → DAI _n _PB _{xx} _I DAI _n _PB _{xx} _O → SPTx_xx_I
S/PDIF Tx/Rx	No	SPDIF _n _TX_O → SPDIF _n _RX_I	SPDIF _n _TX_O → DAI _n _PB _{xx} _I DAI _n _PB _{xx} _O → SPDIF _n _RX_I
SRC	No	SRC _x _DAT_OP_O → SRC _x _DAT_IP_I	SRC _x _DAT_OP_O → DAI _n _PB _{xx} _I DAI _n _PB _{xx} _O → SRC _x _DAT_IP_I

DAI Sources Overview

The following tables provide information of the various DAI sources sorted by group. The shaded cells in the tables denote connection to the other DAI.

DAI0 Group A – Clock Routing Source Signals

The following table lists Group A clock sources.

Table 22-10: DAI0 Group A – Clock Routing

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_PB01_O	Pin Buffer 1
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12
01100 (0xC)	Reserved	
01101 (0xD)	Reserved	
01110 (0xE)	DAI0_CRS_PB03_O	Pin Buffer 3 (DAI1)
01111 (0xF)	DAI0_CRS_PB05_O	Pin Buffer 5 (DAI1)
10000 (0x10)	Reserved	
10001 (0x11)	Reserved	
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20
10100 (0x14)	SPT0_ACLK_O	SPORT 0 Clock A
10101 (0x15)	SPT0_BCLK_O	SPORT 0 Clock B
10110 (0x16)	SPT1_ACLK_O	SPORT 1 Clock A
10111 (0x17)	SPT1_BCLK_O	SPORT 1 Clock B
11000 (0x18)	SPT2_ACLK_O	SPORT 2 Clock A
11001 (0x19)	SPT2_BCLK_O	SPORT 2 Clock B
11010 (0x1A)	SPDIF0_RX_CLK_O	SPDIF 0 Receive Clock Output
11011 (0x1B)	SPDIF0_RX_TDMCLK_O	SPDIF 0 Receive TDM Clock Output

Table 22-10: DAI0 Group A – Clock Routing (Continued)

Selection Code	Source Signal	Description (Source Selection)
11100 (0x1C)	PCG0_CLKA_O	Precision Clock A Output
11101 (0x1D)	PCG0_CLKB_O	Precision Clock B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI0 Group B – Serial Data Source Signals

The group B data sources are listed in the following table. The group B data destinations are configured using [DAI_PBEN0](#) through [DAI_PBEN3](#).

Table 22-11: DAI0 Group B – Serial Data Signals

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI0_PB01_O	Pin Buffer 1
000001 (0x1)	DAI0_PB02_O	Pin Buffer 2
000010 (0x2)	DAI0_PB03_O	Pin Buffer 3
000011 (0x3)	DAI0_PB04_O	Pin Buffer 4
000100 (0x4)	DAI0_PB05_O	Pin Buffer 5
000101 (0x5)	DAI0_PB06_O	Pin Buffer 6
000110 (0x6)	DAI0_PB07_O	Pin Buffer 7
000111 (0x7)	DAI0_PB08_O	Pin Buffer 8
001000 (0x8)	DAI0_PB09_O	Pin Buffer 9
001001 (0x9)	DAI0_PB10_O	Pin Buffer 10
001010 (0xA)	DAI0_PB11_O	Pin Buffer 11
001011 (0xB)	DAI0_PB12_O	Pin Buffer 12
001100 (0xC)	Reserved	
001101 (0xD)	Reserved	
001110 (0xE)	Reserved	
001111 (0xF)	Reserved	
010000 (0x10)	Reserved	
010001 (0x11)	Reserved	
010010 (0x12)	DAI0_PB19_O	Pin Buffer 19
010011 (0x13)	DAI0_PB20_O	Pin Buffer 20
010100 (0x14)	SPT0_AD0_O	SPORT 0 Data AD0
010101 (0x15)	SPT0_AD1_O	SPORT 0 Data AD1
010110 (0x16)	SPT0_BD0_O	SPORT 0 Data BD0
010111 (0x17)	SPT0_BD1_O	SPORT 0 Data BD1
011000 (0x18)	SPT1_AD0_O	SPORT 1 Data AD0
011001 (0x19)	SPT1_AD1_O	SPORT 1 Data AD1
011010 (0x1A)	SPT1_BD0_O	SPORT 1 Data BD0
011011 (0x1B)	SPT1_BD1_O	SPORT 1 Data BD1

Table 22-11: DAI0 Group B – Serial Data Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
011100 (0x1C)	SPT2_AD0_O	SPORT 2 Data AD0
011101 (0x1D)	SPT2_AD1_O	SPORT 2 Data AD1
011110 (0x1E)	SPT2_BD0_O	SPORT 2 Data BD0
011111 (0x1F)	SPT2_BD1_O	SPORT 2 Data BD1
100000 (0x20)	SRC0_DAT_OP_O	SRC0 Data Out
100001 (0x21)	SRC1_DAT_OP_O	SRC1 Data Out
100010 (0x22)	SRC2_DAT_OP_O	SRC2 Data Out
100011 (0x23)	SRC3_DAT_OP_O	SRC3 Data Out
100100 (0x24)	SRC0_TDM_IP_O	SRC0 Data Out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 Data Out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 Data Out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 Data Out
101000 (0x28)	SPDIF0_RX_DAT_O	SPDIF 0 RX Serial Data Out
101100 (0x2C)	SPT3_AD0_O	SPORT 3 Data AD0
101101 (0x2D)	SPT3_AD1_O	SPORT 3 Data AD1
101110 (0x2E)	SPT3_BD0_O	SPORT 3 Data BD0
101111 (0x2F)	SPT3_BD1_O	SPORT 3 Data BD1
110000 (0x30)	SPDIF0_TX_O	SPDIF 0 TX Biphase Stream
110001 (0x31)	SRC7_CRS_DAT_OP_O	SRC7 data out (DAI1)
110010 (0x32)	SRC7_CRS_TDM_IP_O	SRC7 data out (DAI1)
110011 (0x33)–111101 (0x3D)	Reserved	
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

DAI0 Group C – Frame Sync Source Signals

The group C frame sync signal sources are listed in the following table.

Table 22-12: DAI0 Group C – Frame Sync Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_PB01_O	Pin Buffer 1
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3

Table 22-12: DAI0 Group C – Frame Sync Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12
01100 (0xC)	Reserved	
01101 (0xD)	Reserved	
01110 (0xE)	DAI0_CRS_PB04_O	Pin Buffer 4 (DAI1)
01111 (0xF)	DAI0_CRS_PB06_O	Pin Buffer 6 (DAI1)
10000 (0x10)	Reserved	
10001 (0x11)	Reserved	
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20
10100 (0x14)	SPT0_AFS_O	SPORT 0 Frame Sync A
10101 (0x15)	SPT0_BFS_O	SPORT 0 Frame Sync B
10110 (0x16)	SPT1_AFS_O	SPORT 1 Frame Sync A
10111 (0x17)	SPT1_BFS_O	SPORT 1 Frame Sync B
11000 (0x18)	SPT2_AFS_O	SPORT 2 Frame Sync A
11001 (0x19)	SPT2_BFS_O	SPORT 2 Frame Sync B
11010 (0x1A)	SPDIF0_FS_O	SPDIF 0 RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG0_FSA_O	Precision Frame Sync A Output
11101 (0x1D)	PCG0_FSB_O	Precision Frame Sync B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI0 Group D – Pin Signal Assignment Source Signals

The group D pin signal assignment sources are listed in the following table.

Table 22-13: DAI0 Group D – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI0_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI0_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI0_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI0_PB04_O	Pin Buffer 4
0000100 (0x4)	DAI0_PB05_O	Pin Buffer 5
0000101 (0x5)	DAI0_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI0_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI0_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI0_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI0_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI0_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI0_PB12_O	Pin Buffer 12
0001100 (0xC)	Reserved	
0001101 (0xD)	Reserved	
0001110 (0xE)	Reserved	
0001111 (0xF)	Reserved	
0010000 (0x10)	Reserved	
0010001 (0x11)	Reserved	
0010010 (0x12)	DAI0_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI0_PB20_O	Pin Buffer 20
0010100 (0x14)	SPT0_AD0_O	SPORT 0 Data AD0
0010101 (0x15)	SPT0_AD1_O	SPORT 0 Data AD1
0010110 (0x16)	SPT0_BD0_O	SPORT 0 Data BD0
0010111 (0x17)	SPT0_BD1_O	SPORT 0 Data BD1
0011000 (0x18)	SPT1_AD0_O	SPORT 1 Data AD0
0011001 (0x19)	SPT1_AD1_O	SPORT 1 Data AD1
0011010 (0x1A)	SPT1_BD0_O	SPORT 1 Data BD0
0011011 (0x1B)	SPT1_BD1_O	SPORT 1 Data BD1
0011100 (0x1C)	SPT2_AD0_O	SPORT 2 Data AD0
0011101 (0x1D)	SPT2_AD1_O	SPORT 2 Data AD1
0011110 (0x1E)	SPT2_BD0_O	SPORT 2 Data BD0

Table 22-13: DAI0 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0011111 (0x1F)	SPT2_BD1_O	SPORT 2 Data BD1
0100000 (0x20)	SPT0_ACLK_O	SPORT 0 Clock A
0100001 (0x21)	SPT0_BCLK_O	SPORT 0 Clock B
0100010 (0x22)	SPT1_ACLK_O	SPORT 1 Clock A
0100011 (0x23)	SPT1_BCLK_O	SPORT 1 Clock B
0100100 (0x24)	SPT2_ACLK_O	SPORT 2 Clock A
0100101 (0x25)	SPT2_BCLK_O	SPORT 2 Clock B
0100110 (0x26)	SPT0_AFS_O	SPORT 0 Frame Sync A
0100111 (0x27)	SPT0_BFS_O	SPORT 0 Frame Sync B
0101000 (0x28)	SPT1_AFS_O	SPORT 1 Frame Sync A
0101001 (0x29)	SPT1_BFS_O	SPORT 1 Frame Sync B
0101010 (0x2A)	SPT2_AFS_O	SPORT 2 Frame Sync A
0101011 (0x2B)	SPT2_BFS_O	SPORT 2 Frame Sync B
0101100 (0x2C)	SPT3_AD0_O	SPORT 3 Data AD0
0101101 (0x2D)	SPT3_AD1_O	SPORT 3 Data AD1
0101110 (0x2E)	SPT3_BD0_O	SPORT 3 Data BD0
0101111 (0x2F)	SPT3_BD1_O	SPORT 3 Data BD1
0110000 (0x30)	MLB0_CLKOUT	MLB PLL clock output
0110001 (0x31)	SPDIF0_TX_BLKSTART_O	SPDIF 0 TX Block Start Output
0110100 (0x34)	SPT3_ACLK_O	SPORT 3 Clock A
0110101 (0x35)	SPT3_BCLK_O	SPORT 3 Clock B
0110110 (0x36)	SPT3_AFS_O	SPORT 3 Frame Sync A
0110111 (0x37)	SPT3_BFS_O	SPORT 3 Frame Sync B
0111000 (0x38)	PCG0_CLKA_O	Precision Clock A
0111001 (0x39)	PCG0_CLKB_O	Precision Clock B
0111010 (0x3A)	PCG0_FSA_O	Precision Frame Sync A
0111011 (0x3B)	PCG0_FSB_O	Precision Frame Sync B
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 Data Output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 Data Output
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 Data Output

Table 22-13: DAI0 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 Data Output
1000001 (0x41)	SPDIF0_RX_DAT_O	SPDIF 0 RX Data Output
1000010 (0x42)	SPDIF0_FS_O	SPDIF 0 RX Frame Sync Output
1000011 (0x43)	SPDIF0_RXCLK_O	SPDIF 0 RX Clock Output
1000100 (0x44)	SPDIF0_RX_TDMCLK_O	SPDIF 0 RX TDM Clock Output
1000101 (0x45)	SPDIF0_TX_O	SPDIF 0 TX Biphase Encoded Data Output
1000110 (0x46)	SPT0_ATDV_O	SPORT0 Transmit A Data Valid Output
1000111 (0x47)	SPT0_BTDTV_O	SPORT0 Transmit B Data Valid Output
1001000 (0x48)	SPT1_ATDV_O	SPORT1 Transmit A Data Valid Output
1001001 (0x49)	SPT1_BTDTV_O	SPORT1 Transmit B Data Valid Output
1001010 (0x4A)	SPT2_ATDV_O	SPORT2 Transmit A Data Valid Output
1001011 (0x4B)	SPT2_BTDTV_O	SPORT2 Transmit B Data Valid Output
1001100 (0x4C)	SPT3_ATDV_O	SPORT3 Transmit A Data Valid Output
1001101 (0x4D)	SPT3_BTDTV_O	SPORT3 Transmit B Data Valid Output
1001110 (0x4E)	DAI0_CRS_PB05_0	Pin Buffer 5 (DAI1)
1001111 (0x4F)	DAI0_CRS_PB06_0	Pin Buffer 6 (DAI1)
1010000 (0x50)	PCG0_CRS_CLKC_O	Precision Clock C (DAI1)
1010001 (0x51)	PCG0_CRS_CLKD_O	Precision Clock D (DAI1)
1010010 (0x52)	PCG0_CRS_FSC_O	Precision Frame Sync C (DAI1)
1010011 (0x53)	PCG0_CRS_FSD_O	Precision Frame Sync D (DAI1)
1010100 (0x54)	DAI0_CRS_PB03_O	Pin Buffer 3 (DAI1)
1010101 (0x55)	DAI0_CRS_PB04_O	Pin Buffer 4 (DAI1)
1010110 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)
1111111 (0x7F)	HIGH	Logic Level High (1)

DAI0 Group E – Miscellaneous Source Signals

The following table lists the group E miscellaneous signal sources.

Table 22-14: DAI0 Group E – Miscellaneous Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11 Output
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12 Output
01100 (0xC)	Reserved	
01101 (0xD)	Reserved	
01110 (0xE)	Reserved	
01111 (0xF)	Reserved	
10000 (0x10)	Reserved	
10001 (0x11)	Reserved	
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20 Output
10100 (0x14)	SPT0_AFS_O	SPORT 0 Frame Sync A
10101 (0x15)	SPT0_BFS_O	SPORT 0 Frame Sync B
10110 (0x16)	SPT1_AFS_O	SPORT 1 Frame Sync A
10111 (0x17)	SPT1_BFS_O	SPORT 1 Frame Sync B
11000 (0x18)	SPT2_AFS_O	SPORT 2 Frame Sync A
11001 (0x19)	SPT2_BFS_O	SPORT 2 Frame Sync B
11010 (0x1A)	SPDIF0_TX_BLKSTART_O	SPDIF 0 TX Block Start Output
11011 (0x1B)	PCG0_FSA_O	Precision Frame Sync A
11100 (0x1C)	PCG0_CLKB_O	Precision Clock B
11101 (0x1D)	PCG0_FSB_O	Precision Frame Sync B
11110 (0x1E)	LOW	Logic Level Low (0) as a Source

Table 22-14: DAI0 Group E – Miscellaneous Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

DAI0 Group F – Pin Output Enable Source Signals

The group F pin output enable signal sources are listed in the following table. The group F destinations are configured using `DAI_PBEN0` to `DAI_PBEN2` registers.

Table 22-15: DAI0 Group F – Pin Output Enable

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DAI0_MISCA0_O	DAI0 Miscellaneous Control A0 Output
000011 (0x3)	DAI0_MISCA1_O	DAI0 Miscellaneous Control A1 Output
000100 (0x4)	DAI0_MISCA2_O	DAI0 Miscellaneous Control A2 Output
000101 (0x5)	DAI0_MISCA3_O	DAI0 Miscellaneous Control A3 Output
000110 (0x6)	DAI0_MISCA4_O	DAI0 Miscellaneous Control A4 Output
000111 (0x7)	DAI0_MISCA5_O	DAI0 Miscellaneous Control A5 Output
001000 (0x8)	SPT0_ACLK_PBEN_O	SPORT 0 Clock A Output Enable
001001 (0x9)	SPT0_AFS_PBEN_O	SPORT 0 Frame Sync A Output Enable
001010 (0xA)	SPT0_AD0_PBEN_O	SPORT 0 Data AD0 Output Enable
001011 (0xB)	SPT0_AD1_PBEN_O	SPORT 0 Data AD1 Output Enable
001100 (0xC)	SPT0_BCLK_PBEN_O	SPORT 0 Clock B Output Enable
001101 (0xD)	SPT0_BFS_PBEN_O	SPORT 0 Frame Sync B Output Enable
001110 (0xE)	SPT0_BD0_PBEN_O	SPORT 0 Data BD0 Output Enable
001111 (0xF)	SPT0_BD1_PBEN_O	SPORT 0 Data BD1 Output Enable
010000 (0x10)	SPT1_ACLK_PBEN_O	SPORT 1 Clock A Output Enable
010001 (0x11)	SPT1_AFS_PBEN_O	SPORT 1 Frame Sync A Output Enable
010010 (0x12)	SPT1_AD0_PBEN_O	SPORT 1 Data AD0 Output Enable
010011 (0x13)	SPT1_AD1_PBEN_O	SPORT 1 Data AD1 Output Enable
010100 (0x14)	SPT1_BCLK_PBEN_O	SPORT 1 Clock B Output Enable
010101 (0x15)	SPT1_BFS_PBEN_O	SPORT 1 Frame Sync B Output Enable
010110 (0x16)	SPT1_BD0_PBEN_O	SPORT 1 Data BD0 Output Enable
010111 (0x17)	SPT1_BD1_PBEN_O	SPORT 1 Data BD1 Output Enable
011000 (0x18)	SPT2_ACLK_PBEN_O	SPORT 2 Clock A Output Enable
011001 (0x19)	SPT2_AFS_PBEN_O	SPORT 2 Frame Sync A Output Enable
011010 (0x1A)	SPT2_AD0_PBEN_O	SPORT 2 Data AD0 Output Enable
011011 (0x1B)	SPT2_AD1_PBEN_O	SPORT 2 Data AD1 Output Enable

Table 22-15: DAI0 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
011100 (0x1C)	SPT2_BCLK_PBEN_O	SPORT 2 Clock B Output Enable
011101 (0x1D)	SPT2_BFS_PBEN_O	SPORT 2 Frame Sync B Output Enable
011110 (0x1E)	SPT2_BD0_PBEN_O	SPORT 2 Data BD0 Output Enable
011111 (0x1F)	SPT2_BD1_PBEN_O	SPORT 2 Data BD1 Output Enable
100000 (0x20)	SPT3_ACLK_PBEN_O	SPORT 3 Clock A Output Enable
100001 (0x21)	SPT3_AFS_PBEN_O	SPORT 3 Frame Sync A Output Enable
100010 (0x22)	SPT3_AD0_PBEN_O	SPORT 3 Data AD0 Output Enable
100011 (0x23)	SPT3_AD1_PBEN_O	SPORT 3 Data AD1 Output Enable
100100 (0x24)	SPT3_BCLK_PBEN_O	SPORT 2 Clock B Output Enable
100101 (0x25)	SPT3_BFS_PBEN_O	SPORT 3 Frame Sync B Output Enable
100110 (0x26)	SPT3_BD0_PBEN_O	SPORT 3 Data BD0 Output Enable
100111 (0x27)	SPT3_BD1_PBEN_O	SPORT 3 Data BD1 Output Enable
101000 (0x28)	SPT0_ATDV_PBEN_O	SPORT 0 A Transmit Data Valid Output
101001 (0x29)	SPT0_BTDTV_PBEN_O	SPORT 0 B Transmit Data Valid Output
101010 (0x2A)	SPT1_ATDV_PBEN_O	SPORT 1 A Transmit Data Valid Output
101011 (0x2B)	SPT1_BTDTV_PBEN_O	SPORT 1 B Transmit Data Valid Output
101100 (0x2C)	SPT2_ATDV_PBEN_O	SPORT 2 A Transmit Data Valid Output
101101 (0x2D)	SPT2_BTDTV_PBEN_O	SPORT 2 B Transmit Data Valid Output
100111 (0x2E)	SPT3_ATDV_PBEN_O	SPORT 3 A Transmit Data Valid Output
101110 (0x2F)	SPT3_BTDTV_PBEN_O	SPORT 3 B Transmit Data Valid Output
110000 (0x30)–1111111 (0x3F)	Reserved	

DAI1 Group A – Clock Routing Source Signals

The group A clock routing signal sources are listed in the following table.

Table 22-16: DAI1 Group A – Clock Routing Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI1_PB01_O	Pin Buffer 1
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5

Table 22-16: DAI1 Group A – Clock Routing Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12
01100 (0xC)	Reserved	
01101 (0xD)	Reserved	
01110 (0xE)	DAI1_CRS_PB03_O	Pin Buffer 3 (DAI0)
01111 (0xF)	DAI1_CRS_PB05_O	Pin Buffer 5 (DAI0)
10000 (0x10)	Reserved	
10001 (0x11)	Reserved	
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20
10100 (0x14)	SPT4_ACLK_O	SPORT 4 Clock A
10101 (0x15)	SPT4_BCLK_O	SPORT 4 Clock B
10110 (0x16)	SPT5_ACLK_O	SPORT 5 Clock A
10111 (0x17)	SPT5_BCLK_O	SPORT 5 Clock B
11000 (0x18)	SPT6_ACLK_O	SPORT 6 Clock A
11001 (0x19)	SPT6_BCLK_O	SPORT 6 Clock B
11010 (0x1A)	SPDIF1_RX_CLK_O	SPDIF 1 Receive Clock Output
11011 (0x1B)	SPDIF1_RX_TDMCLK_O	SPDIF 1 Receive TDM Clock Output
11100 (0x1C)	PCG0_CLKC_O	Precision Clock C Output
11101 (0x1D)	PCG0_CLKD_O	Precision Clock D Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI1 Group B – Serial Data Source Signals

The group B data sources are listed in the following table.

Table 22-17: DAI1 Group B – Serial Data Signals

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI1_PB01_O	Pin Buffer 1
000001 (0x1)	DAI1_PB02_O	Pin Buffer 2
000010 (0x2)	DAI1_PB03_O	Pin Buffer 3
000011 (0x3)	DAI1_PB04_O	Pin Buffer 4
000100 (0x4)	DAI1_PB05_O	Pin Buffer 5
000101 (0x5)	DAI1_PB06_O	Pin Buffer 6
000110 (0x6)	DAI1_PB07_O	Pin Buffer 7
000111 (0x7)	DAI1_PB08_O	Pin Buffer 8
001000 (0x8)	DAI1_PB09_O	Pin Buffer 9
001001 (0x9)	DAI1_PB10_O	Pin Buffer 10
001010 (0xA)	DAI1_PB11_O	Pin Buffer 11
001011 (0xB)	DAI1_PB12_O	Pin Buffer 12
001100 (0xC) -010001 (0x11)	Reserved	
010010 (0x12)	DAI1_PB19_O	Pin Buffer 19
010011 (0x13)	DAI1_PB20_O	Pin Buffer 20
010100 (0x14)	SPT4_AD0_O	SPORT 4 Data AD0
010101 (0x15)	SPT4_AD1_O	SPORT 4 Data AD1
010110 (0x16)	SPT4_BD0_O	SPORT 4 Data BD0
010111 (0x17)	SPT4_BD1_O	SPORT 4 Data BD1
011000 (0x18)	SPT5_AD0_O	SPORT 5 Data AD0
011001 (0x19)	SPT5_AD1_O	SPORT 5 Data AD1
011010 (0x1A)	SPT5_BD0_O	SPORT 5 Data BD0
011011 (0x1B)	SPT5_BD1_O	SPORT 5 Data BD1
011100 (0x1C)	SPT6_AD0_O	SPORT 6 Data AD0
011101 (0x1D)	SPT6_AD1_O	SPORT 6 Data AD1
011110 (0x1E)	SPT6_BD0_O	SPORT 6 Data BD0
011111 (0x1F)	SPT6_BD1_O	SPORT 6 Data BD1
100000 (0x20)	SRC4_DAT_OP_O	SRC4 Data Out
100001 (0x21)	SRC5_DAT_OP_O	SRC5 Data Out
100010 (0x22)	SRC6_DAT_OP_O	SRC6 Data Out
100011 (0x23)	SRC7_DAT_OP_O	SRC7 Data Out

Table 22-17: DAI1 Group B – Serial Data Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
100100 (0x24)	SRC4_TDM_IP_O	SRC4 Data Out
100101 (0x25)	SRC5_TDM_IP_O	SRC5 Data Out
100110 (0x26)	SRC6_TDM_IP_O	SRC6 Data Out
100111 (0x27)	SRC7_TDM_IP_O	SRC7 Data Out
101000 (0x28)	SPDIF1_RX_DAT_O	SPDIF 1 RX Serial Data Out
101100(0x2C)	SPT7_AD0_O	SPORT 7 Data AD0
101101(0x2D)	SPT7_AD1_O	SPORT 7 Data AD1
101110(0x2E)	SPT7_BD0_O	SPORT 7 Data BD0
101111(0x2F)	SPT7_BD1_O	SPORT 7 Data BD1
110000(0x30)	SPDIF1_TX_O	SPDIF 1 TX Biphase Stream
110001 (0x31)	SRC3_CRS_DAT_OP_O	SRC3 Data Out (DAI0)
110010 (0x32)	SRC3_CRS_TDM_IP_O	SRC3 Data Out (DAI0)
110001(0x33)–111101(0x3D)	Reserved	
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

DAI1 Group C – Frame Sync Source Signals

The group C frame sync signals are listed in the following table.

Table 22-18: DAI1 Group C – Frame Sync Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_PB01_O	Pin Buffer 1
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11

Table 22-18: DAI1 Group C – Frame Sync Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12
01100 (0xC)	Reserved	
01101 (0xD)	Reserved	
01110 (0xE)	DAI1_CRS_PB04_O	Pin Buffer 4 (DAI0)
01111 (0xF)	DAI1_CRS_PB06_O	Pin Buffer 6 (DAI0)
10000 (0x10)	Reserved	
10001 (0x11)	Reserved	
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20
10100 (0x14)	SPT4_AFS_O	SPORT 4 Frame Sync A
10101 (0x15)	SPT4_BFS_O	SPORT 4 Frame Sync B
10110 (0x16)	SPT5_AFS_O	SPORT 5 Frame Sync A
10111 (0x17)	SPT5_BFS_O	SPORT 5 Frame Sync B
11000 (0x18)	SPT6_AFS_O	SPORT 6 Frame Sync A
11001 (0x19)	SPT6_BFS_O	SPORT 6 Frame Sync B
11010 (0x1A)	SPDIF1_FS_O	SPDIF 1 RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG0_FSC_O	Precision Frame Sync C Output
11101 (0x1D)	PCG0_FSD_O	Precision Frame Sync D Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI1 Group D – Pin Signal Assignment Source Signals

The group D pin signal assignments are listed in the following table.

Table 22-19: DAI1 Group D – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI1_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI1_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI1_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI1_PB04_O	Pin Buffer 4
0000100 (0x4)	DAI1_PB05_O	Pin Buffer 5

Table 22-19: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0000101 (0x5)	DAI1_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI1_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI1_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI1_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI1_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI1_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI1_PB12_O	Pin Buffer 12
0001100 (0xC) - 0010001 (0x11)	Reserved	
0010010 (0x12)	DAI1_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI1_PB20_O	Pin Buffer 20
0010100 (0x14)	SPT4_AD0_O	SPORT 4 Data AD0
0010101 (0x15)	SPT4_AD1_O	SPORT 4 Data AD1
0010110 (0x16)	SPT4_BD0_O	SPORT 4 Data BD0
0010111 (0x17)	SPT4_BD1_O	SPORT 4 Data BD1
0011000 (0x18)	SPT5_AD0_O	SPORT 5 Data AD0
0011001 (0x19)	SPT5_AD1_O	SPORT 5 Data AD1
0011010 (0x1A)	SPT5_BD0_O	SPORT 5 Data BD0
0011011 (0x1B)	SPT5_BD1_O	SPORT 5 Data BD1
0011100 (0x1C)	SPT6_AD0_O	SPORT 6 Data AD0
0011101 (0x1D)	SPT6_AD1_O	SPORT 6 Data AD1
0011110 (0x1E)	SPT6_BD0_O	SPORT 6 Data BD0
0011111 (0x1F)	SPT6_BD1_O	SPORT 6 Data BD1
0100000 (0x20)	SPT4_ACLK_O	SPORT 4 Clock A
0100001 (0x21)	SPT4_BCLK_O	SPORT4 Clock B
0100010 (0x22)	SPT5_ACLK_O	SPORT 5 Clock A
0100011 (0x23)	SPT5_BCLK_O	SPORT 5 Clock B
0100100 (0x24)	SPT6_ACLK_O	SPORT 6 Clock A
0100101 (0x25)	SPT6_BCLK_O	SPORT 6 Clock B
0100110 (0x26)	SPT4_AFS_O	SPORT 4 Frame Sync A
0100111 (0x27)	SPT4_BFS_O	SPORT 4 Frame Sync B
0101000 (0x28)	SPT5_AFS_O	SPORT 5 Frame Sync A

Table 22-19: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0101001 (0x29)	SPT5_BFS_O	SPORT 5 Frame Sync B
0101010 (0x2A)	SPT6_AFS_O	SPORT 5 Frame Sync A
0101011 (0x2B)	SPT6_BFS_O	SPORT 5 Frame Sync B
0101100 (0x2C)	SPT7_AD0_O	SPORT 7 Data AD0
0101101 (0x2D)	SPT7_AD1_O	SPORT 7 Data AD1
0101110 (0x2E)	SPT7_BD0_O	SPORT 7 Data BD0
0101111 (0x2F)	SPT7_BD1_O	SPORT 7 Data BD1
0110000 (0x30)	MLB0_CLKOUT	MLB PLL clock output
0110001 (0x31)	SPDIF1_TX_BLKSTART_O	SPDIF 1 TX Block Start Output
0110100 (0x34)	SPT7_ACLK_O	SPORT 7 Clock A
0110101 (0x35)	SPT7_BCLK_O	SPORT 7 Clock B
0110110 (0x36)	SPT7_AFS_O	SPORT 7 Frame Sync A
0110111 (0x37)	SPT7_BFS_O	SPORT 7 Frame Sync B
0111000 (0x38)	PCG0_CLKC_O	Precision Clock C
0111001 (0x39)	PCG0_CLKD_O	Precision Clock D
0111010 (0x3A)	PCG0_FSC_O	Precision Frame Sync C
0111011 (0x3B)	PCG0_FSD_O	Precision Frame Sync D
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC4_DAT_OP_O	SRC4 Data Output
0111110 (0x3E)	SRC5_DAT_OP_O	SRC5 Data Output
0111111 (0x3F)	SRC6_DAT_OP_O	SRC6 Data Output
1000000 (0x40)	SRC7_DAT_OP_O	SRC7 Data Output
1000001 (0x41)	SPDIF1_RX_DAT_O	SPDIF 1 RX Data Output
1000010 (0x42)	SPDIF1_FS_O	SPDIF 1 RX Frame Sync Output
1000011 (0x43)	SPDIF1_RXCLK_O	SPDIF 1 RX Clock Output
1000100 (0x44)	SPDIF1_RX_TDMCLK_O	SPDIF 1 RX TDM Clock Output
1000101 (0x45)	SPDIF1_TX_O	SPDIF 1 TX Biphase Encoded Data Output
1000110 (0x46)	SPT4_ATDV_O	SPORT4 Transmit A Data Valid Output
1000111 (0x47)	SPT4_BTDTV_O	SPORT4 Transmit B Data Valid Output
1001000 (0x48)	SPT5_ATDV_O	SPORT5 Transmit A Data Valid Output
1001001 (0x49)	SPT5_BTDTV_O	SPORT5 Transmit B Data Valid Output

Table 22-19: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
1001010 (0x4A)	SPT6_ATDV_O	SPORT6 Transmit A Data Valid Output
1001011 (0x4B)	SPT6_BTDTV_O	SPORT6 Transmit B Data Valid Output
1001100 (0x4C)	SPT7_ATDV_O	SPORT7 Transmit A Data Valid Output
1001101 (0x4D)	SPT7_BTDTV_O	SPORT7 Transmit B Data Valid Output
1001110 (0x4E)	DAI1_CRS_PB05_O	Pin Buffer 5 (DAI0)
1001111 (0x4F)	DAI1_CRS_PB06_O	Pin Buffer 6 (DAI0)
1010000 (0x50)	PCG0_CRS_CLKC_O	Precision Clock C (DAI0)
1010001 (0x51)	PCG0_CRS_CLKD_O	Precision Clock D (DAI0)
1010010 (0x52)	PCG0_CRS_FSC_O	Precision Frame Sync C (DAI0)
1010011 (0x53)	PCG0_CRS_FSD_O	Precision Frame Sync D (DAI0)
1010100 (0x54)	DAI1_CRS_PB03_O	Pin Buffer 3 (DAI0)
1010101 (0x55)	DAI1_CRS_PB04_O	Pin Buffer 4 (DAI0)
1010110 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)
1111111 (0x7F)	HIGH	Logic Level High (1)

DAI1 Group E – Miscellaneous Source Signals

The group E miscellaneous signals are listed in the following table.

Table 22-20: DAI1 Group E – Miscellaneous Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI1_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11 Output

Table 22-20: DAI1 Group E – Miscellaneous Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12 Output
01100 (0xC) - 10001 (0x11)	Reserved	
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20 Output
10100 (0x14)	SPT4_AFS_O	SPORT 4 Frame Sync A
10101 (0x15)	SPT4_BFS_O	SPORT 4 Frame Sync B
10110 (0x16)	SPT5_AFS_O	SPORT 5 Frame Sync A
10111 (0x17)	SPT5_BFS_O	SPORT 5 Frame Sync B
11000 (0x18)	SPT6_AFS_O	SPORT 6 Frame Sync A
11001 (0x19)	SPT6_BFS_O	SPORT 6 Frame Sync B
11010 (0x1A)	SPDIF1_TX_BLKSTART_O	SPDIF 1 TX Block Start Output
11011 (0x1B)	PCG0_FSC_O	Precision Frame Sync C
11100 (0x1C)	PCG0_CLKD_O	Precision Clock D
11101 (0x1D)	PCG0_FSD_O	Precision Frame Sync D
11110 (0x1E)	LOW	Logic Level Low (0) as a Source
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

DAI1 Group F – Pin Output Enable Source Signals

The group F pin output enable signals are listed in the following table.

Table 22-21: DAI1 Group F – Pin Output Enable

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DAI1_MISCA0_O	DAI1 Miscellaneous Control A0 Output
000011 (0x3)	DAI1_MISCA1_O	DAI1 Miscellaneous Control A1 Output
000100 (0x4)	DAI1_MISCA2_O	DAI1 Miscellaneous Control A2 Output
000101 (0x5)	DAI1_MISCA3_O	DAI1 Miscellaneous Control A3 Output
000110 (0x6)	DAI1_MISCA4_O	DAI1 Miscellaneous Control A4 Output
000111 (0x7)	DAI1_MISCA5_O	DAI1 Miscellaneous Control A5 Output
001000 (0x8)	SPT4_ACLK_PBEN_O	SPORT 4 Clock A Output Enable
001001 (0x9)	SPT4_AFS_PBEN_O	SPORT 4 Frame Sync A Output Enable

Table 22-21: DAI1 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
001010 (0xA)	SPT4_AD0_PBEN_O	SPORT 4 Data AD0 Output Enable
001011 (0xB)	SPT4_AD1_PBEN_O	SPORT 4 Data AD1 Output Enable
001100 (0xC)	SPT4_BCLK_PBEN_O	SPORT 4 Clock B Output Enable
001101 (0xD)	SPT4_BFS_PBEN_O	SPORT 4 Frame Sync B Output Enable
001110 (0xE)	SPT4_BD0_PBEN_O	SPORT 4 Data BD0 Output Enable
001111 (0xF)	SPT4_BD1_PBEN_O	SPORT 4 Data BD1 Output Enable
010000 (0x10)	SPT5_ACLK_PBEN_O	SPORT 5 Clock A Output Enable
010001 (0x11)	SPT5_AFS_PBEN_O	SPORT 5 Frame Sync A Output Enable
010010 (0x12)	SPT5_AD0_PBEN_O	SPORT 5 Data AD0 Output Enable
010011 (0x13)	SPT5_AD1_PBEN_O	SPORT 5 Data AD1 Output Enable
010100 (0x14)	SPT5_BCLK_PBEN_O	SPORT 5 Clock B Output Enable
010101 (0x15)	SPT5_BFS_PBEN_O	SPORT 5 Frame Sync B Output Enable
010110 (0x16)	SPT5_BD0_PBEN_O	SPORT 5 Data BD0 Output Enable
010111 (0x17)	SPT5_BD1_PBEN_O	SPORT 5 Data BD1 Output Enable
011000 (0x18)	SPT6_ACLK_PBEN_O	SPORT 6 Clock A Output Enable
011001 (0x19)	SPT6_AFS_PBEN_O	SPORT 6 Frame Sync A Output Enable
011010 (0x1A)	SPT6_AD0_PBEN_O	SPORT 6 Data AD0 Output Enable
011011 (0x1B)	SPT6_AD1_PBEN_O	SPORT 6 Data AD1 Output Enable
011100 (0x1C)	SPT6_BCLK_PBEN_O	SPORT 6 Clock B Output Enable
011101 (0x1D)	SPT6_BFS_PBEN_O	SPORT 6 Frame Sync B Output Enable
011110 (0x1E)	SPT6_BD0_PBEN_O	SPORT 6 Data BD0 Output Enable
011111 (0x1F)	SPT6_BD1_PBEN_O	SPORT 6 Data BD1 Output Enable
100000 (0x20)	SPT7_ACLK_PBEN_O	SPORT 7 Clock A Output Enable
100001 (0x21)	SPT7_AFS_PBEN_O	SPORT 7 Frame Sync A Output Enable
100010 (0x22)	SPT7_AD0_PBEN_O	SPORT 7 Data AD0 Output Enable
100011 (0x23)	SPT7_AD1_PBEN_O	SPORT 7 Data AD1 Output Enable
100100 (0x24)	SPT7_BCLK_PBEN_O	SPORT 6 Clock B Output Enable
100101 (0x25)	SPT7_BFS_PBEN_O	SPORT 7 Frame Sync B Output Enable
100110 (0x26)	SPT7_BD0_PBEN_O	SPORT 7 Data BD0 Output Enable
100111 (0x27)	SPT7_BD1_PBEN_O	SPORT 7 Data BD1 Output Enable
101000 (0x28)	SPT4_ATDV_PBEN_O	SPORT 4 A Transmit Data Valid Output

Table 22-21: DAI1 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
101001 (0x29)	SPT4_BTDTV_PBEN_O	SPORT 4 B Transmit Data Valid Output
101010 (0x2A)	SPT5_ATDV_PBEN_O	SPORT 5 A Transmit Data Valid Output
101011 (0x2B)	SPT5_BTDTV_PBEN_O	SPORT 5 B Transmit Data Valid Output
101100 (0x2C)	SPT6_ATDV_PBEN_O	SPORT 6 A Transmit Data Valid Output
101101 (0x2D)	SPT6_BTDTV_PBEN_O	SPORT 6 B Transmit Data Valid Output
100111 (0x2E)	SPT7_ATDV_PBEN_O	SPORT 7 A Transmit Data Valid Output
101110 (0x2F)	SPT7_BTDTV_PBEN_O	SPORT 7 B Transmit Data Valid Output
110000 (0x30)–1111111 (0x3F)	Reserved	

DAI Destination Registers Overview

The tables in [DAI Routing Capabilities](#) provide high level descriptions that illustrate source (output) connections to destinations (inputs) depending on the routing groups A through F. This sections lists the various input fields (INx), which are described in detail in the [ADSP-2156x DAI Register Descriptions](#) section.

Table 22-22: Clock Destination Registers (Group A)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_CLK0	IN0	SPT0_ACLK_I	SPT4_ACLK_I	SPORT0A or SPORT4A Clock
	IN1	SPT0_BCLK_I	SPT4_BCLK_I	SPORT0B or SPORT4B Clock
	IN2	SPT1_ACLK_I	SPT5_ACLK_I	SPORT1A or SPORT5A Clock
	IN3	SPT1_BCLK_I	SPT5_BCLK_I	SPORT1B or SPORT5B Clock
	IN4	SPT2_ACLK_I	SPT6_ACLK_I	SPORT2A or SPORT6A Clock
	IN5	SPT2_BCLK_I	SPT6_BCLK_I	SPORT2B or SPORT6B Clock
DAI_CLK1	IN0	SRC0_CLK_IP_I	SRC4_CLK_IP_I	SRC0 or SRC4 Clock Input
	IN1	SRC0_CLK_OP_I	SRC4_CLK_OP_I	SRC0 or SRC4 Clock Output
	IN2	SRC1_CLK_IP_I	SRC5_CLK_IP_I	SRC1 or SRC5 Clock Input
	IN3	SRC1_CLK_OP_I	SRC5_CLK_OP_I	SRC1 or SRC5 Clock Output
	IN4	SRC2_CLK_IP_I	SRC6_CLK_IP_I	SRC2 or SRC6 Clock Input
	IN5	SRC2_CLK_OP_I	SRC6_CLK_OP_I	SRC2 or SRC6 Clock Output
DAI_CLK2	IN0	SRC3_CLK_IP_I	SRC7_CLK_IP_I	SRC3 or SRC7 Clock Input
	IN1	SRC3_CLK_OP_I	SRC7_CLK_OP_I	SRC3 or SRC7 Clock Output
	IN2	SPDIF0_TX_CLK_I	SPDIF1_TX_CLK_I	SPDIF0 or SPDIF1 TX Clock

Table 22-22: Clock Destination Registers (Group A) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_CLK3	IN5	SPDIF0_TX_HFCLK_I	SPDIF1_TX_HFCLK_I	SPDIF0 or SPDIF1 TX HF Clock
DAI_CLK4	IN0	PCG0_EXTCLKA_I	PCG0_EXTCLKC_I	PCG0 External Clock A or C
	IN1	PCG0_EXTCLKB_I	PCG0_EXTCLKD_I	PCG0 External Clock B or D
	IN3	SPDIF0_TX_EXT_SYNC_I	SPDIF1_TX_EXT_SYNC_I	SPDIF0 or SPDIF1 TX External Sync
	IN4	PCG0_SYNC_CLKA_I	PCG0_SYNC_CLKC_I	PCG0 Sync Clock A or C
	IN5	PCG0_SYNC_CLKB_I	PCG0_SYNC_CLKD_I	PCG0 Sync Clock B or D
DAI_CLK5	IN0	SPT3_ACLK_I	SPT7_ACLK_I	SPORT3A or SPORT 7A Clock
	IN1	SPT3_BCLK_I	SPT7_BCLK_I	SPORT3B or SPORT 7B Clock

Table 22-23: Data Destination Registers (Group B)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_DAT0	IN0	SPT0_AD0_I	SPT4_AD0_I	SPORT0A or SPORT4A Primary Data
	IN1	SPT0_AD1_I	SPT4_AD1_I	SPORT0A or SPORT4A Secondary Data
	IN2	SPT0_BD0_I	SPT4_BD0_I	SPORT0B or SPORT4B Primary Data
	IN3	SPT0_BD1_I	SPT4_BD1_I	SPORT0B or SPORT4B Secondary Data
	IN4	SPT1_AD0_I	SPT5_AD0_I	SPORT1A or SPORT5A Primary Data
DAI_DAT1	IN0	SPT1_AD1_I	SPT5_AD1_I	SPORT1A or SPORT5A Secondary Data
	IN1	SPT1_BD0_I	SPT5_BD0_I	SPORT1B or SPORT5B Primary Data
	IN2	SPT1_BD1_I	SPT5_BD1_I	SPORT1B or SPORT5B Secondary Data
	IN3	SPT2_AD0_I	SPT6_AD0_I	SPORT2A or SPORT6A Primary Data
	IN4	SPT2_AD1_I	SPT6_AD1_I	SPORT2A or SPORT6A Secondary Data
DAI_DAT2	IN0	SPT2_BD0_I	SPT6_BD0_I	SPORT2B or SPORT6B Primary Data
	IN1	SPT2_BD1_I	SPT6_BD1_I	SPORT2B or SPORT6B Secondary Data
	IN2	SRC0_DAT_IP_I	SRC4_DAT_IP_I	SRC0 or SRC4 Data
	IN3	SRC1_DAT_IP_I	SRC5_DAT_IP_I	SRC1 or SRC5 Data
	IN4	SRC2_DAT_IP_I	SRC6_DAT_IP_I	SRC2 or SRC6 Data

Table 22-23: Data Destination Registers (Group B) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_DAT3	IN0	SRC3_DAT_IP_I	SRC7_DAT_IP_I	SRC3 or SRC7 Data
	IN1	SRC0_DAT_TDM_OP_I	SRC4_DAT_TDM_OP_I	SRC0 or SRC4 TDM output port
	IN2	SRC1_DAT_TDM_OP_I	SRC5_DAT_TDM_OP_I	SRC1 or SRC5 TDM Output port Data
	IN3	SRC2_DAT_TDM_OP_I	SRC6_DAT_TDM_OP_I	SRC2 or SRC6 TDM Output port Data
	IN4	SRC3_DAT_TDM_OP_I	SRC7_DAT_TDM_OP_I	SRC3 or SRC7 TDM Output port Data
DAI_DAT4	IN0	SPDIF0_TX_DAT_I	SPDIF1_TX_DAT_I	SPDIF0 or SPDIF1 serial transmitter Data
DAI_DAT5	IN4	SPDIF0_RX_I	SPDIF1_RX_I	SPDIF0 or SPDIF1 receiver bi-phase Data
DAI_DAT6	IN0	SPT3_AD0_I	SPT7_AD0_I	SPORT3A or SPORT7A Primary Data
	IN1	SPT3_AD1_I	SPT7_AD1_I	SPORT3A or SPORT7A Secondary Data
	IN2	SPT3_BD0_I	SPT7_BD0_I	SPORT3B or SPORT7B Primary Data
	IN3	SPT3_BD1_I	SPT7_BD1_I	SPORT3B or SPORT7B Secondary Data
	IN4	Reserved	Reserved	

Table 22-24: Frame Sync Destination Registers (Group C)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_FS0	IN0	SPT0_AFS_I	SPT4_AFS_I	SPORT0A or SPORT4A Frame Sync
	IN1	SPT0_BFS_I	SPT4_BFS_I	SPORT0B or SPORT4B Frame Sync
	IN2	SPT1_AFS_I	SPT5_AFS_I	SPORT1A or SPORT5A Frame Sync
	IN3	SPT1_BFS_I	SPT5_BFS_I	SPORT1B or SPORT5B Frame Sync
	IN4	SPT2_AFS_I	SPT6_AFS_I	SPORT2A or SPORT6A Frame Sync
	IN5	SPT2_BFS_I	SPT6_BFS_I	SPORT2B or SPORT6B Frame Sync
DAI_FS1	IN0	SRC0_FS_IP_I	SRC4_FS_IP_I	SRC0 or SRC4 Frame Sync Input
	IN1	SRC0_FS_OP_I	SRC4_FS_OP_I	SRC0 or SRC4 Frame Sync Output
	IN2	SRC1_FS_IP_I	SRC5_FS_IP_I	SRC1 or SRC5 Frame Sync Input
	IN3	SRC1_FS_OP_I	SRC5_FS_OP_I	SRC1 or SRC5 Frame Sync Output
	IN4	SRC2_FS_IP_I	SRC6_FS_IP_I	SRC2 or SRC6 Frame Sync Input

Table 22-24: Frame Sync Destination Registers (Group C) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN5	SRC2_FS_OP_I	SRC6_FS_OP_I	SRC2 or SRC6 Frame Sync Output
DAI_FS2	IN0	SRC3_FS_IP_I	SRC7_FS_IP_I	SRC3 or SRC7 Frame Sync Input
	IN1	SRC3_FS_OP_I	SRC7_FS_OP_I	SRC3 or SRC7 Frame Sync Output
	IN2	SPDIF0_TX_FS_I	SPDIF1_TX_FS_I	SPDIF0 or SPDIF1 Transmit Frame Sync
DAI_FS4	IN0	SPT3_AFS_I	SPT7_AFS_I	SPORT3A or SPORT7A Frame Sync
	IN1	SPT3_BFS_I	SPT7_BFS_I	SPORT3B or SPORT7B Frame Sync

Table 22-25: Pin Buffer Assignment Destination Registers (Group D)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_PIN0	IN0	DAI0_PB01_I	DAI1_PB01_I	Pin Buffer 1
	IN1	DAI0_PB02_I	DAI1_PB02_I	Pin Buffer 2
	IN2	DAI0_PB03_I	DAI1_PB03_I	Pin Buffer 3
	IN3	DAI0_PB04_I	DAI1_PB04_I	Pin Buffer 4
DAI_PIN1	IN0	DAI0_PB05_I	DAI1_PB05_I	Pin Buffer 5
	IN1	DAI0_PB06_I	DAI1_PB06_I	Pin Buffer 6
	IN2	DAI0_PB07_I	DAI1_PB07_I	Pin Buffer 7
	IN3	DAI0_PB08_I	DAI1_PB08_I	Pin Buffer 8
DAI_PIN2	IN0	DAI0_PB09_I	DAI1_PB09_I	Pin Buffer 9
	IN1	DAI0_PB10_I	DAI1_PB10_I	Pin Buffer 10
	IN2	DAI0_PB11_I	DAI1_PB11_I	Pin Buffer 11
	IN3	DAI0_PB12_I	DAI1_PB12_I	Pin Buffer 12
DAI_PIN3	IN0	Reserved		
	IN1	Reserved		
	IN2	Reserved		
	IN3	Reserved		
DAI_PIN4	IN0	Reserved		
	IN1	Reserved		
	IN2	DAI0_PB19_I	DAI1_PB19_I	Pin Buffer 19
	IN3	DAI0_PB20_I	DAI1_PB20_I	Pin Buffer 20
	IN4	INV_DAI0_PB19_I	INV_DAI1_PB19_I	Inverted Pin Buffer 19

Table 22-25: Pin Buffer Assignment Destination Registers (Group D) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN5	INV_DAI0_PB20_I	INV_DAI1_PB20_I	Inverted Pin Buffer 20

Table 22-26: Pin Buffer Enable Destination Registers (Group E)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_PBEN0	IN0	DAI0_PBEN01_I	DAI1_PBEN01_I	Pin Buffer 1 Enable
	IN1	DAI0_PBEN02_I	DAI1_PBEN02_I	Pin Buffer 2 Enable
	IN2	DAI0_PBEN03_I	DAI1_PBEN03_I	Pin Buffer 3 Enable
	IN3	DAI0_PBEN04_I	DAI1_PBEN04_I	Pin Buffer 4 Enable
	IN4	DAI0_PBEN05_I	DAI1_PBEN05_I	Pin Buffer 5 Enable
DAI_PBEN1	IN0	DAI0_PBEN06_I	DAI1_PBEN06_I	Pin Buffer 6 Enable
	IN1	DAI0_PBEN07_I	DAI1_PBEN07_I	Pin Buffer 7 Enable
	IN2	DAI0_PBEN08_I	DAI1_PBEN08_I	Pin Buffer 8 Enable
	IN3	DAI0_PBEN09_I	DAI1_PBEN09_I	Pin Buffer 9 Enable
	IN4	DAI0_PBEN10_I	DAI1_PBEN10_I	Pin Buffer 10 Enable
DAI_PBEN2	IN0	DAI0_PBEN11_I	DAI1_PBEN11_I	Pin Buffer 11 Enable
	IN1	DAI0_PBEN12_I	DAI1_PBEN12_I	Pin Buffer 12 Enable
	IN2	Reserved		
	IN3	Reserved		
	IN4	Reserved		
DAI_PBEN3	IN0	Reserved		
	IN1	Reserved		
	IN2	Reserved		
	IN3	DAI0_PBEN19_I	DAI1_PBEN19_I	Pin Buffer 19 Enable
	IN4	DAI0_PBEN20_I	DAI1_PBEN20_I	Pin Buffer 20 Enable

The the *Miscellaneous Control Registers* table, DAI_MISCA_x_I is an alias name for DAI_INT_x_I.

Table 22-27: Miscellaneous Control Destination Registers (Group F)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_MISC0	IN6	DAI0_MISCA0_I/ DAI0_INT_6_I	DAI1_MISCA0_I/ DAI1_INT_6_I	DAIx miscellaneous A0 input, or DAIx interrupt 28

Table 22-27: Miscellaneous Control Destination Registers (Group F) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN7	DAI0_MISCA1_I/ DAI0_INT_7_I	DAI1_MISCA1_I/ DAI1_INT_7_I	DAIx miscellaneous A1 input, or DAIx interrupt 29
	IN8	DAI0_MISCA2_I/ DAI0_INT_8_I	DAI1_MISCA2_I/ DAI1_INT_8_I	DAIx miscellaneous A2 input, or DAIx interrupt 30
	IN9	DAI0_MISCA3_I/ DAI0_INT_9_I	DAI1_MISCA3_I/ DAI1_INT_9_I	DAIx miscellaneous A3 input, or DAIx interrupt 31
	IN10	DAI0_MISCA4_I	DAI1_MISCA4_I	DAIx miscellaneous A4 input
	IN11	DAI0_MISCA5_I	DAI1_MISCA5_I	DAIx miscellaneous A5 input
	INV10	DAI0_INV_MISCA4_I	DAI1_INV_MISCA4_I	DAIx inverted miscellaneous A4 input
	INV11	DAI0_INV_MISCA5_I	DAI1_INV_MISCA5_I	DAIx inverted miscellaneous A5 input
DAI_MISC1	IN0	DAI0_INT_0	DAI1_INT_0	DAIx interrupt 22
	IN1	DAI0_INT_1	DAI1_INT_1	DAIx interrupt 23
	IN2	DAI0_INT_2	DAI1_INT_2	DAIx interrupt 24
	IN3	DAI0_INT_3	DAI1_INT_3	DAIx interrupt 25
	IN4	DAI0_INT_4	DAI1_INT_4	DAIx interrupt 26
	IN5	DAI0_INT_5	DAI1_INT_5	DAIx interrupt 27

The *Interrupt Events* table presents the information for the following registers.

- DAI_IMSK_PRI (Core Interrupt Priority Assignment) register
- DAI_IMSK_FE (Falling-Edge Interrupt Mask) and DAI_IMSK_RE (Rising-Edge Interrupt Mask) registers
- DAI_IRPTL_H (High-Priority Interrupt Latch) and DAI_IRPTL_L (Low-Priority Interrupt Latch) registers
- DAI_IRPTL_HS (Shadow High Interrupt Latch) and DAI_IRPTL_LS (Shadow Low Interrupt Latch) registers

Table 22-28: Interrupt Events

Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
RXVALID	SPDIF0_RXVALID_INT	SPDIF1_RXVALID_INT	SPDIF0 or SPDIF1 RX Valid interrupt
RXLOCK	SPDIF0_RXLOCK_INT	SPDIF1_RXLOCK_INT	SPDIF0 or SPDIF1 RX Lock interrupt
RXLOSSOF-LOCK	SPDIF0_RXLOSSOFLOCK_INT	SPDIF1_RXLOSSOFLOCK_INT	SPDIF0 or SPDIF1 RX Loss of Lock interrupt

Table 22-28: Interrupt Events (Continued)

Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
RXNONAUDIO	SPDIF0_RXNONAUDIO_INT	SPDIF1_RXNONAUDIO_INT	SPDIF0 or SPDIF1 RX Non-audio interrupt
SRC0MUTE	SRC0_MUTE_INT	SRC4_MUTE_INT	DAI0 or DAI1 ASRC0 Mute interrupt
SRC1MUTE	SRC1_MUTE_INT	SRC5_MUTE_INT	DAI0 or DAI1 ASRC1 Mute interrupt
SRC2MUTE	SRC2_MUTE_INT	SRC6_MUTE_INT	DAI0 or DAI1 ASRC2 Mute interrupt
SRC3MUTE	SRC3_MUTE_INT	SRC7_MUTE_INT	DAI0 or DAI1 ASRC3 Mute interrupt
MISCINT0	DAI0_INT_00	DAI1_INT_00	DAI0 or DAI1 Miscellaneous Interrupt 0
MISCINT1	DAI0_INT_01	DAI1_INT_01	DAI0 or DAI1 Miscellaneous Interrupt 1
MISCINT2	DAI0_INT_02	DAI1_INT_02	DAI0 or DAI1 Miscellaneous Interrupt 2
MISCINT3	DAI0_INT_03	DAI1_INT_03	DAI0 or DAI1 Miscellaneous Interrupt 3
MISCINT4	DAI0_INT_04	DAI1_INT_04	DAI0 or DAI1 Miscellaneous Interrupt 4
MISCINT5	DAI0_INT_05	DAI1_INT_05	DAI0 or DAI1 Miscellaneous Interrupt 5
MISCINT6	DAI0_EXTMISCA0_INT/ DAI0_INT_06	DAI1_EXTMISCA0_INT/ DAI1_INT_06	DAI0 or DAI1 Miscellaneous Interrupt 6 / External Miscellaneous A0 Interrupt
MISCINT7	DAI0_EXTMISCA1_INT/ DAI0_INT_07	DAI1_EXTMISCA1_INT/ DAI1_INT_07	DAI0 or DAI1 Miscellaneous Interrupt 7 / External Miscellaneous A1 Interrupt
MISCINT8	DAI0_EXTMISCA2_INT/ DAI0_INT_08	DAI1_EXTMISCA2_INT/ DAI1_INT_08	DAI0 or DAI1 Miscellaneous Interrupt 8 / External Miscellaneous A2 Interrupt
MISCINT9	DAI0_EXTMISCA3_INT/ DAI0_INT_09	DAI1_EXTMISCA3_INT/ DAI1_INT_09	DAI0 or DAI1 Miscellaneous Interrupt 9 / External Miscellaneous A3 Interrupt

SPORT Grouping with DAI

DAI supports two registers [DAI_GBL_SP_EN](#) and [DAI_GBL_INT_EN](#) to group enabling/disabling and interrupt/trigger generation/reception for multiple SPORTs.

For more information about this, refer to the [Grouping of SPORTs](#) section.

ADSP-2156x DAI Register Descriptions

The Digital Audio Interface (DAI) registers are used to configure DAI destinations for the DAI sources shown in the Group A - F Routing tables. (DAI) contains the following registers.

Table 22-29: ADSP-2156x DAI Register List

Name	Description
DAI_CLK0	Clock Routing Control Register 0
DAI_CLK1	Clock Routing Control Register 1
DAI_CLK2	Clock Routing Control Register 2
DAI_CLK3	Clock Routing Control Register 3
DAI_CLK4	Clock Routing Control Register 4
DAI_CLK5	Clock Routing Control Register 5
DAI_DAT0	Serial Data Routing Control Register 0
DAI_DAT1	Serial Data Routing Control Register 1
DAI_DAT2	Serial Data Routing Control Register 2
DAI_DAT3	Serial Data Routing Control Register 3
DAI_DAT4	Serial Data Routing Control Register 4
DAI_DAT5	Serial Data Routing Control Register 5
DAI_DAT6	Serial Data Routing Control Register 6
DAI_FS0	Frame Sync Routing Control Register 0
DAI_FS1	Frame Sync Routing Control Register 1
DAI_FS2	Frame Sync Routing Control Register 2
DAI_FS4	Frame Sync Routing Control Register 4
DAI_GBL_INT_EN	Global SPORT Interrupt Grouping Register
DAI_GBL_SP_EN	Global SPORT Enable Register
DAI_IMSK_FE	Falling-Edge Interrupt Mask Register
DAI_IMSK_PRI	Core Interrupt Priority Assignment Register
DAI_IMSK_RE	Rising-Edge Interrupt Mask Register
DAI_IRPTL_H	High Priority Interrupt Latch Register
DAI_IRPTL_HS	Shadow High Priority Interrupt Latch Register
DAI_IRPTL_L	Low Priority Interrupt Latch Register
DAI_IRPTL_LS	Shadow Low Priority Interrupt Latch Register

Table 22-29: ADSP-2156x DAI Register List (Continued)

Name	Description
DAI_MISC0	Miscellaneous Control Register 0
DAI_MISC1	Miscellaneous Control Register 1
DAI_PBEN0	Pin Buffer Enable Register 0
DAI_PBEN1	Pin Buffer Enable Register 1
DAI_PBEN2	Pin Buffer Enable Register 2
DAI_PBEN3	Pin Buffer Enable Register 3
DAI_PIN0	Pin Buffer Assignment Register 0
DAI_PIN1	Pin Buffer Assignment Register 1
DAI_PIN2	Pin Buffer Assignment Register 2
DAI_PIN4	Pin Buffer Assignment Register 4
DAI_PIN_STAT	Pin Status Register

Clock Routing Control Register 0

The `DAI_CLK0` register provides clock routing connections for the serial ports (SPORTs).

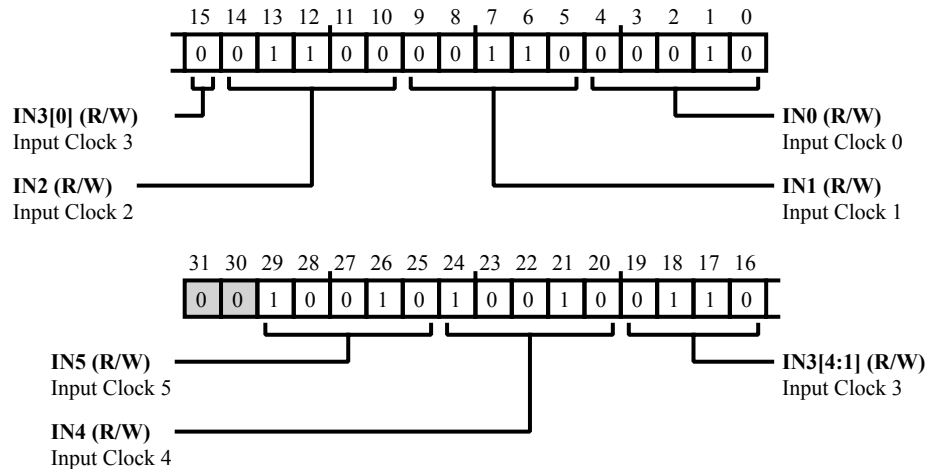


Figure 22-17: DAI_CLK0 Register Diagram

Table 22-30: DAI_CLK0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. DAI_CLK0 . IN5 holds the Source signal assignment that will be routed to the DAI_CLK0 . IN5 Destination. Refer to the Group A Signals table for Source and Destination mappings
24:20 (R/W)	IN4	Input Clock 4. DAI_CLK0 . IN4 holds the Source signal assignment that will be routed to the DAI_CLK0 . IN4 Destination. Refer to the Group A Signals table for Source and Destination mappings
19:15 (R/W)	IN3	Input Clock 3. DAI_CLK0 . IN3 holds the Source signal assignment that will be routed to the DAI_CLK0 . IN3 Destination. Refer to the Group A Signals table for Source and Destination mappings
14:10 (R/W)	IN2	Input Clock 2. DAI_CLK0 . IN2 holds the Source signal assignment that will be routed to the DAI_CLK0 . IN2 Destination. Refer to the Group A Signals table for Source and Destination mappings
9:5 (R/W)	IN1	Input Clock 1. DAI_CLK0 . IN1 holds the Source signal assignment that will be routed to the DAI_CLK0 . IN1 Destination. Refer to the Group A Signals table for Source and Destination mappings

Table 22-30: DAI_CLK0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Clock 0. DAI_CLK0.IN0 holds the Source signal assignment that will be routed to the DAI_CLK0.IN0 Destination. Refer to the Group A Signals table for Source and Destination mappings

Clock Routing Control Register 1

The `DAI_CLK1` register provides clock routing connections for the asynchronous sample rate converters (ASRC).

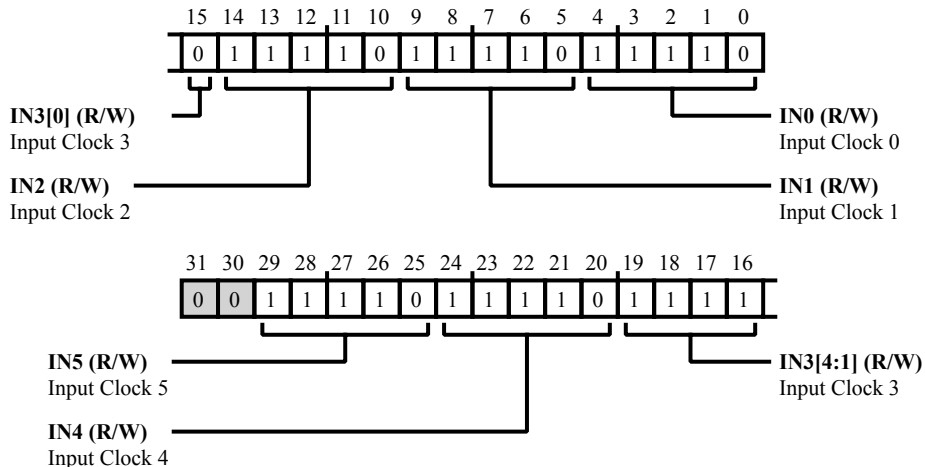


Figure 22-18: DAI_CLK1 Register Diagram

Table 22-31: DAI_CLK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. DAI_CLK1 . IN5 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN5 Destination. Refer to the Group A Signals table for Source and Destination mappings
24:20 (R/W)	IN4	Input Clock 4. DAI_CLK1 . IN4 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN4 Destination. Refer to the Group A Signals table for Source and Destination mappings
19:15 (R/W)	IN3	Input Clock 3. DAI_CLK1 . IN3 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN3 Destination. Refer to the Group A Signals table for Source and Destination mappings
14:10 (R/W)	IN2	Input Clock 2. DAI_CLK1 . IN2 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN2 Destination. Refer to the Group A Signals table for Source and Destination mappings
9:5 (R/W)	IN1	Input Clock 1. DAI_CLK1 . IN1 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN1 Destination. Refer to the Group A Signals table for Source and Destination mappings.

Table 22-31: DAI_CLK1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Clock 0. DAI_CLK1 . IN0 holds the Source signal assignment that will be routed to the DAI_CLK1 . IN0 Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 2

The `DAI_CLK2` register provides clock routing connections for the S/PDIF and ASRC.

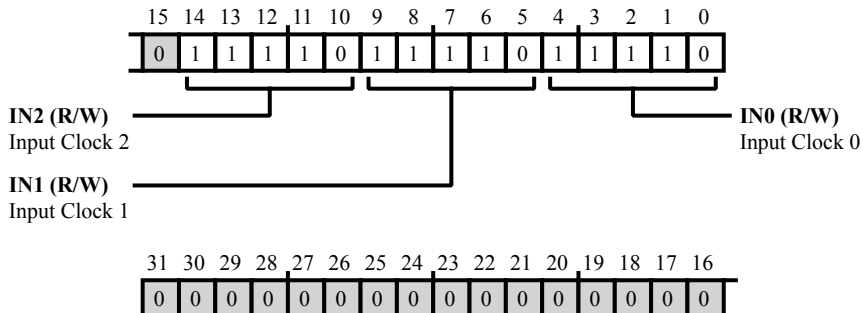


Figure 22-19: DAI_CLK2 Register Diagram

Table 22-32: DAI_CLK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:10 (R/W)	IN2	Input Clock 2. DAI_CLK2 . IN2 holds the Source signal assignment that will be routed to the DAI_CLK2 . IN2 Destination. Refer to the Group A Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Clock 1. DAI_CLK2 . IN1 holds the Source signal assignment that will be routed to the DAI_CLK2 . IN1 Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. DAI_CLK2 . IN0 holds the Source signal assignment that will be routed to the DAI_CLK2 . IN0 Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 3

The `DAI_CLK3` register provides clock routing connections for the S/PDIF.

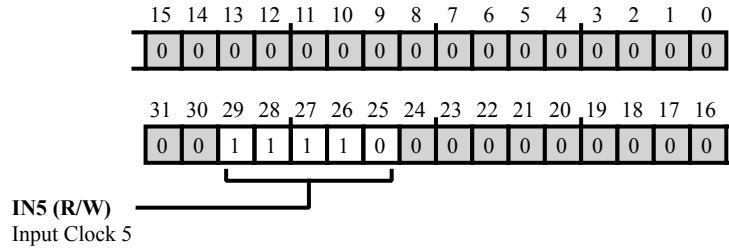


Figure 22-20: DAI_CLK3 Register Diagram

Table 22-33: DAI_CLK3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. The <code>DAI_CLK3.IN5</code> bit field provides the S/PDIF oversampling clock.

Clock Routing Control Register 4

The `DAI_CLK4` register provides clock routing connections for the precision clock generators (PCGs).

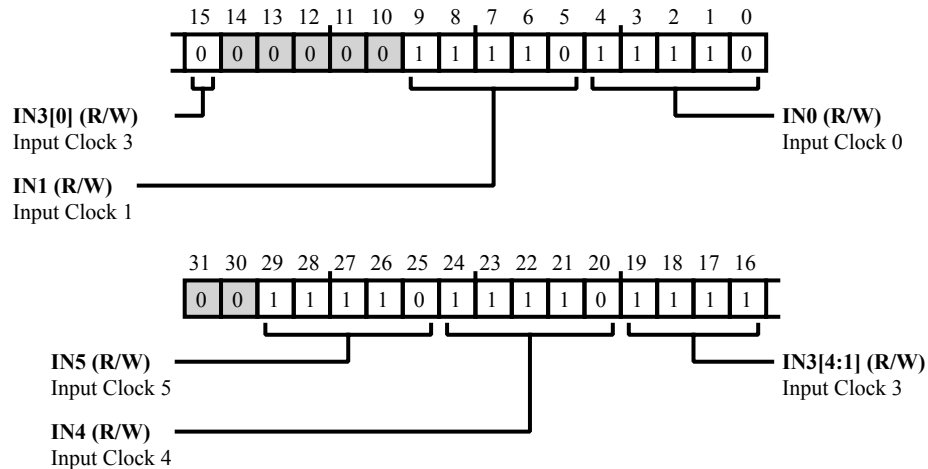


Figure 22-21: DAI_CLK4 Register Diagram

Table 22-34: DAI_CLK4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. DAI_CLK4 . IN5 holds the Source signal assignment that will be routed to the DAI_CLK4 . IN5 Destination. Refer to the Group A Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Clock 4. DAI_CLK4 . IN4 holds the Source signal assignment that will be routed to the DAI_CLK4 . IN4 Destination. Refer to the Group A Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Clock 3. DAI_CLK4 . IN3 holds the Source signal assignment that will be routed to the DAI_CLK4 . IN3 Destination. Refer to the Group A Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Clock 1. DAI_CLK4 . IN1 holds the Source signal assignment that will be routed to the DAI_CLK4 . IN1 Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. DAI_CLK4 . IN0 holds the Source signal assignment that will be routed to the DAI_CLK4 . IN0 Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 5

The `DAI_CLK5` register provides clock routing connections for the serial ports.

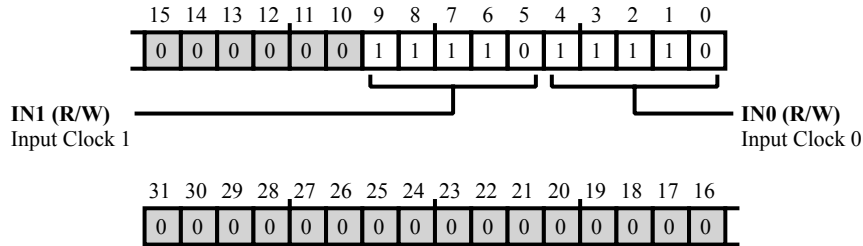


Figure 22-22: DAI_CLK5 Register Diagram

Table 22-35: DAI_CLK5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN1	Input Clock 1. DAI_CLK5.IN1 holds the Source signal assignment that will be routed to the DAI_CLK5.IN1 Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. DAI_CLK5.IN0 holds the Source signal assignment that will be routed to the DAI_CLK5.IN0 Destination. Refer to the Group A Signals table for Source and Destination mappings.

Serial Data Routing Control Register 0

The `DAI_DAT0` register routes serial data to the serial ports.

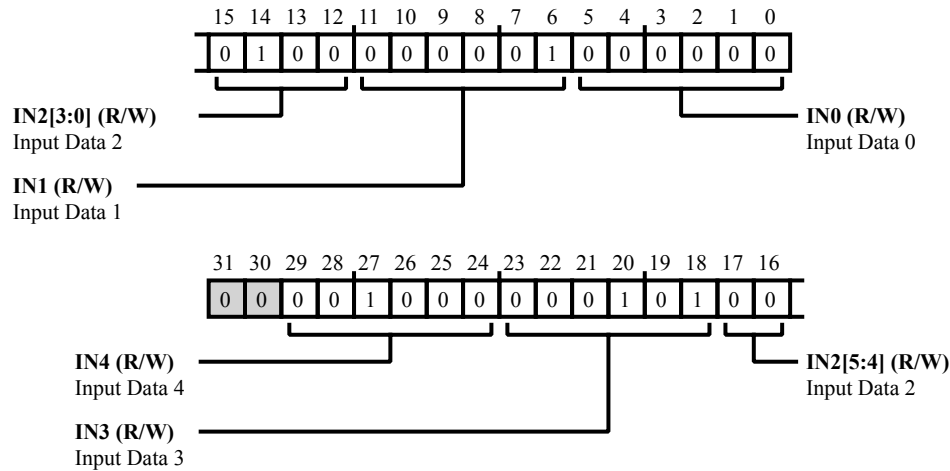


Figure 22-23: DAI_DAT0 Register Diagram

Table 22-36: DAI_DAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. DAI_DAT0 . IN4 holds the Source signal assignment that will be routed to the DAI_DAT0 . IN4 Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. DAI_DAT0 . IN3 holds the Source signal assignment that will be routed to the DAI_DAT0 . IN3 Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. DAI_DAT0 . IN2 holds the Source signal assignment that will be routed to the DAI_DAT0 . IN2 Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. DAI_DAT0 . IN1 holds the Source signal assignment that will be routed to the DAI_DAT0 . IN1 Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. DAI_DAT0 . IN0 holds the Source signal assignment that will be routed to the DAI_DAT0 . IN0 Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 1

The `DAI_DAT1` register routes serial data to the serial ports.

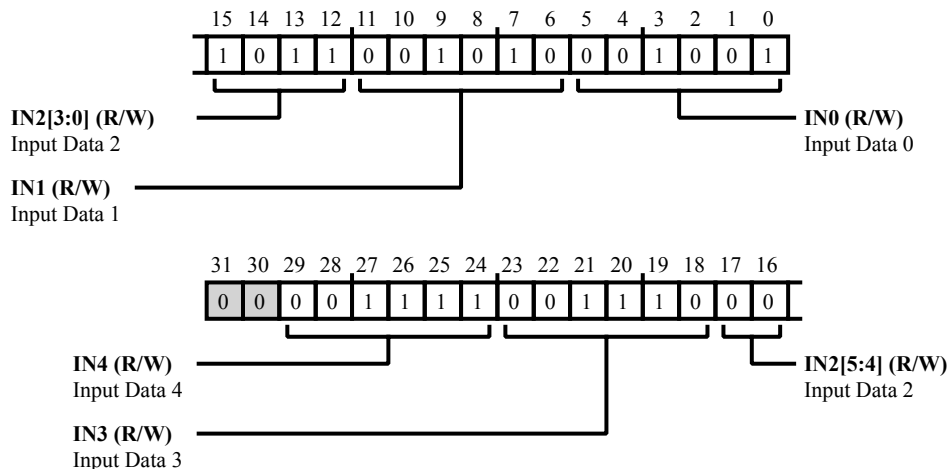


Figure 22-24: DAI_DAT1 Register Diagram

Table 22-37: DAI_DAT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. DAI_DAT1 . IN4 holds the Source signal assignment that will be routed to the DAI_DAT1 . IN4 Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. DAI_DAT1 . IN3 holds the Source signal assignment that will be routed to the DAI_DAT1 . IN3 Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. DAI_DAT1 . IN2 holds the Source signal assignment that will be routed to the DAI_DAT1 . IN2 Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. DAI_DAT1 . IN1 holds the Source signal assignment that will be routed to the DAI_DAT1 . IN1 Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. DAI_DAT1 . IN0 holds the Source signal assignment that will be routed to the DAI_DAT1 . IN0 Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 2

The `DAI_DAT2` register routes serial data to the serial ports and the asynchronous sample rate converter (ASRC).

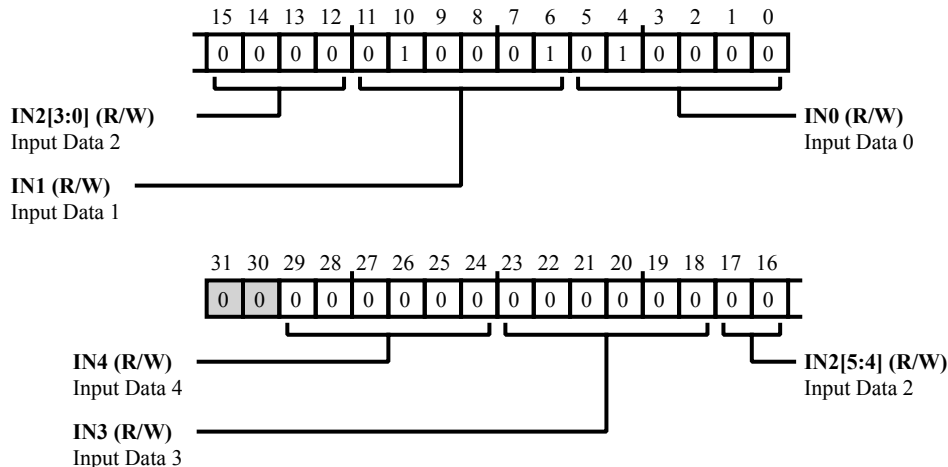


Figure 22-25: DAI_DAT2 Register Diagram

Table 22-38: DAI_DAT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. DAI_DAT2 . IN4 holds the Source signal assignment that will be routed to the DAI_DAT2 . IN4 Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. DAI_DAT2 . IN3 holds the Source signal assignment that will be routed to the DAI_DAT2 . IN3 Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. DAI_DAT2 . IN2 holds the Source signal assignment that will be routed to the DAI_DAT2 . IN2 Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. DAI_DAT2 . IN1 holds the Source signal assignment that will be routed to the DAI_DAT2 . IN1 Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. DAI_DAT2 . IN0 holds the Source signal assignment that will be routed to the DAI_DAT2 . IN0 Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 3

The `DAI_DAT3` register routes serial data to the asynchronous sample rate converter (ASRC).

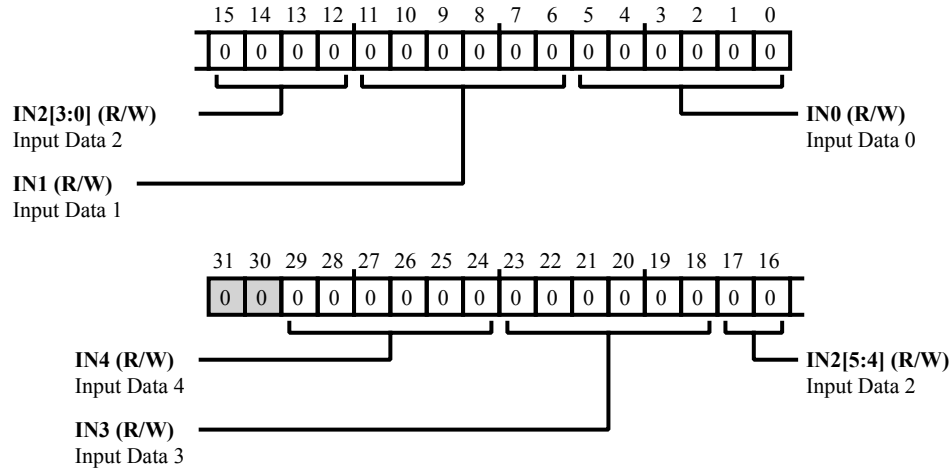


Figure 22-26: DAI_DAT3 Register Diagram

Table 22-39: DAI_DAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. DAI_DAT3 . IN4 holds the Source signal assignment that will be routed to the DAI_DAT3 . IN4 Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. DAI_DAT3 . IN3 holds the Source signal assignment that will be routed to the DAI_DAT3 . IN3 Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. DAI_DAT3 . IN2 holds the Source signal assignment that will be routed to the DAI_DAT3 . IN2 Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. DAI_DAT3 . IN1 holds the Source signal assignment that will be routed to the DAI_DAT3 . IN1 Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. DAI_DAT3 . IN0 holds the Source signal assignment that will be routed to the DAI_DAT3 . IN0 Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 4

The `DAI_DAT4` register routes serial data to the S/PDIF.

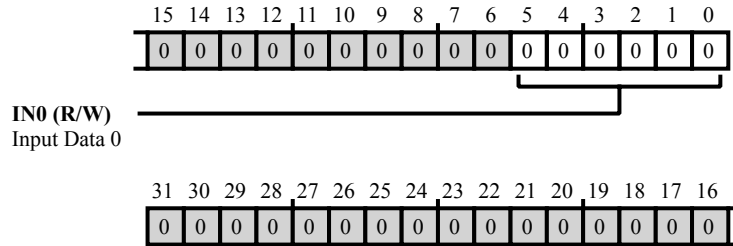


Figure 22-27: `DAI_DAT4` Register Diagram

Table 22-40: `DAI_DAT4` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	IN0	Input Data 0. <code>DAI_DAT4.IN0</code> holds the Source signal assignment that will be routed to the <code>DAI_DAT4.IN0</code> Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 5

The `DAI_DAT5` register routes serial data to the S/PDIF.

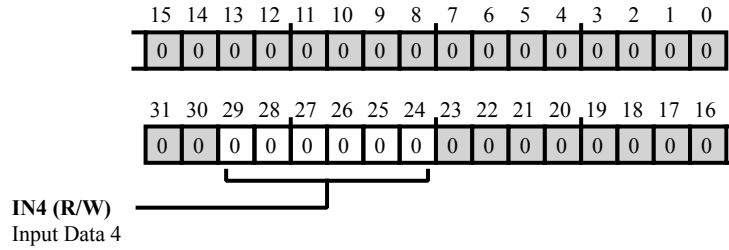


Figure 22-28: DAI_DAT5 Register Diagram

Table 22-41: DAI_DAT5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. The <code>DAI_DAT5</code> . IN4 bit field routes input data to the S/PDIF Biphase receiver stream.

Serial Data Routing Control Register 6

The `DAI_DAT6` register routes serial data to the serial ports.

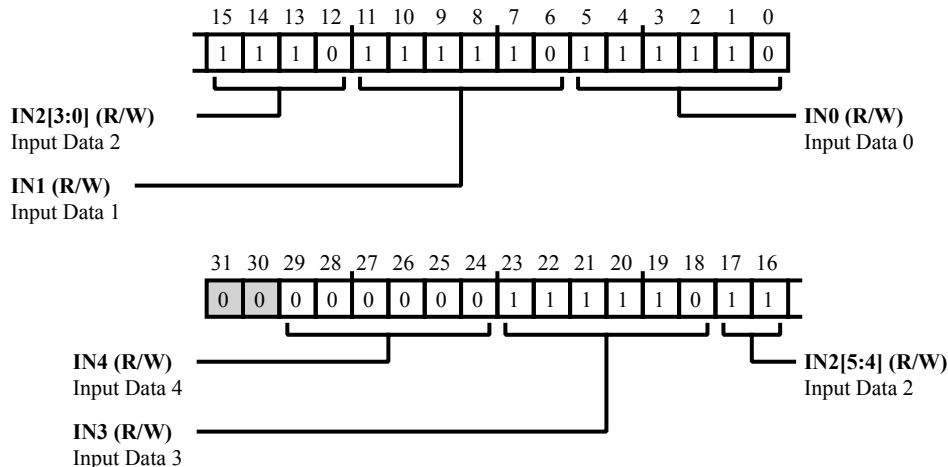


Figure 22-29: DAI_DAT6 Register Diagram

Table 22-42: DAI_DAT6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. DAI_DAT6 . IN4 holds the Source signal assignment that will be routed to the DAI_DAT6 . IN4 Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. DAI_DAT6 . IN3 holds the Source signal assignment that will be routed to the DAI_DAT6 . IN3 Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. DAI_DAT6 . IN2 holds the Source signal assignment that will be routed to the DAI_DAT6 . IN2 Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. DAI_DAT6 . IN1 holds the Source signal assignment that will be routed to the DAI_DAT6 . IN1 Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. DAI_DAT6 . IN0 holds the Source signal assignment that will be routed to the DAI_DAT6 . IN0 Destination. Refer to the Group B Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 0

The `DAI_FS0` register routes frame syncs to the serial ports.

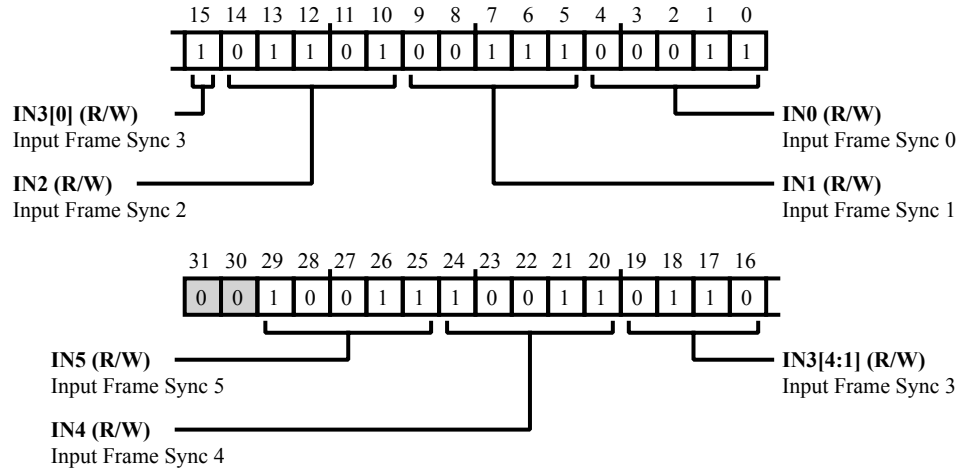


Figure 22-30: DAI_FS0 Register Diagram

Table 22-43: DAI_FS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Frame Sync 5. DAI_FS0 . IN5 holds the Source signal assignment that will be routed to the DAI_FS0 . IN5 Destination. Refer to the Group C Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Frame Sync 4. DAI_FS0 . IN4 holds the Source signal assignment that will be routed to the DAI_FS0 . IN4 Destination. Refer to the Group C Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Frame Sync 3. DAI_FS0 . IN3 holds the Source signal assignment that will be routed to the DAI_FS0 . IN3 Destination. Refer to the Group C Signals table for Source and Destination mappings.
14:10 (R/W)	IN2	Input Frame Sync 2. DAI_FS0 . IN2 holds the Source signal assignment that will be routed to the DAI_FS0 . IN2 Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. DAI_FS0 . IN1 holds the Source signal assignment that will be routed to the DAI_FS0 . IN1 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Table 22-43: DAI_FS0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Frame Sync 0. DAI_FS0.IN0 holds the Source signal assignment that will be routed to the DAI_FS0.IN0 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 1

The `DAI_FS1` register routes frame syncs to the asynchronous sample rate converter (ASRC).

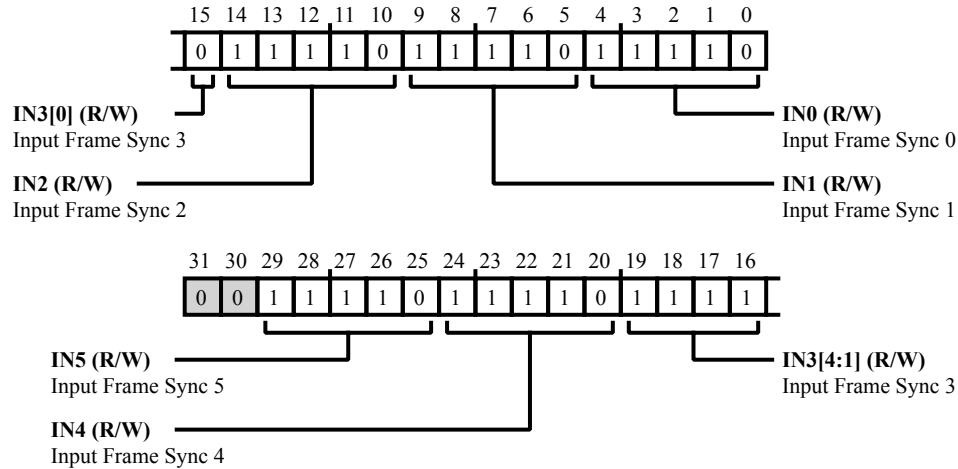


Figure 22-31: DAI_FS1 Register Diagram

Table 22-44: DAI_FS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Frame Sync 5. DAI_FS1 . IN5 holds the Source signal assignment that will be routed to the DAI_FS1 . IN5 Destination. Refer to the Group C Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Frame Sync 4. DAI_FS1 . IN4 holds the Source signal assignment that will be routed to the DAI_FS1 . IN4 Destination. Refer to the Group C Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Frame Sync 3. DAI_FS1 . IN3 holds the Source signal assignment that will be routed to the DAI_FS1 . IN3 Destination. Refer to the Group C Signals table for Source and Destination mappings.
14:10 (R/W)	IN2	Input Frame Sync 2. DAI_FS1 . IN2 holds the Source signal assignment that will be routed to the DAI_FS1 . IN2 Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. DAI_FS1 . IN1 holds the Source signal assignment that will be routed to the DAI_FS1 . IN1 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Table 22-44: DAI_FS1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Frame Sync 0. DAI_FS1 . IN0 holds the Source signal assignment that will be routed to the DAI_FS1 . IN0 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 2

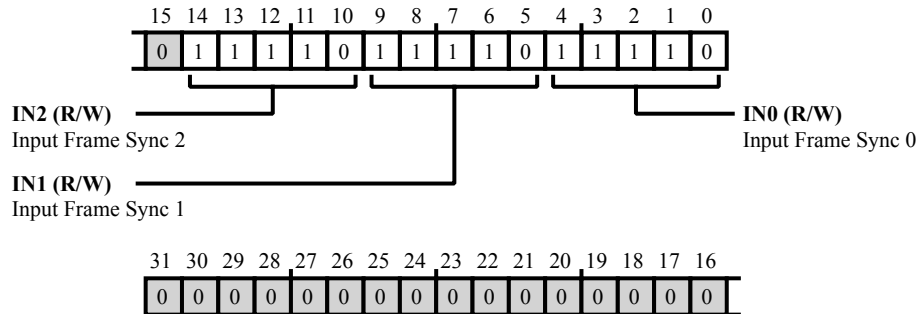


Figure 22-32: DAI_FS2 Register Diagram

Table 22-45: DAI_FS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:10 (R/W)	IN2	Input Frame Sync 2. DAI_FS2 . IN2 holds the Source signal assignment that will be routed to the DAI_FS2 . IN2 Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. The DAI_FS2 . IN1 bit field routes the frame sync output to the ASRC3 frame sync input port.
4:0 (R/W)	IN0	Input Frame Sync 0. DAI_FS2 . IN0 holds the Source signal assignment that will be routed to the DAI_FS2 . IN0 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 4

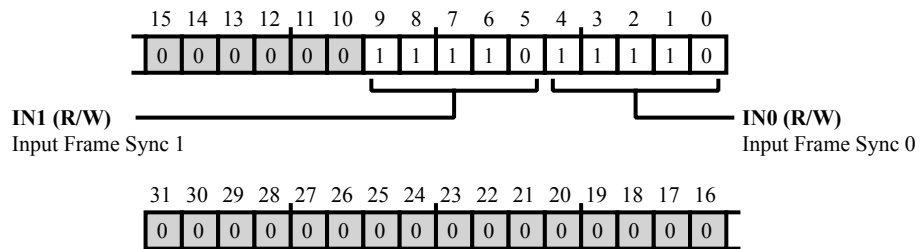


Figure 22-33: DAI_FS4 Register Diagram

Table 22-46: DAI_FS4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN1	Input Frame Sync 1. DAI_FS4 . IN1 holds the Source signal assignment that will be routed to the DAI_FS4 . IN1 Destination. Refer to the Group C Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Frame Sync 0. DAI_FS4 . IN0 holds the Source signal assignment that will be routed to the DAI_FS4 . IN0 Destination. Refer to the Group C Signals table for Source and Destination mappings.

Global SPORT Interrupt Grouping Register

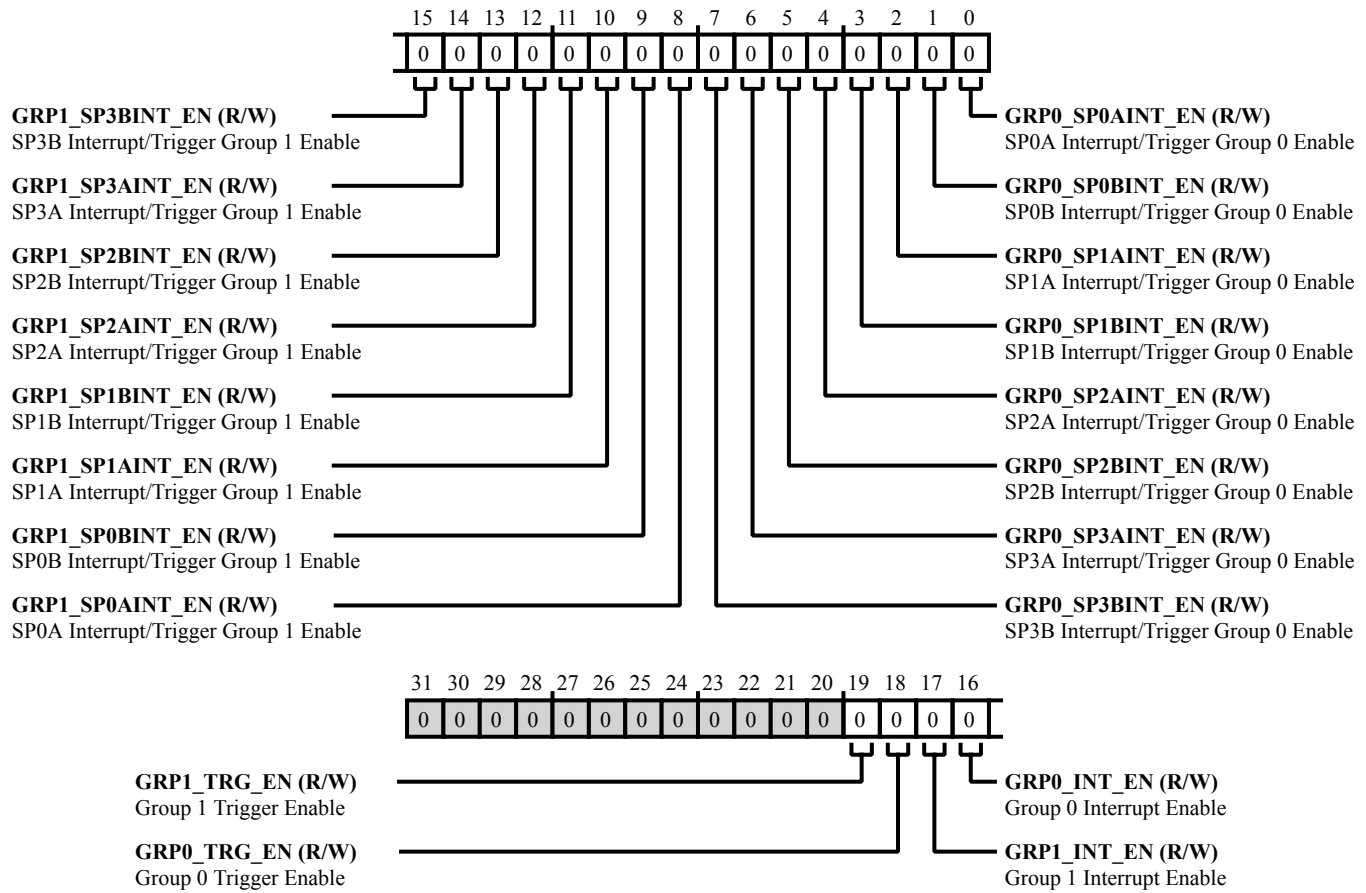


Figure 22-34: DAI_GBL_INT_EN Register Diagram

Table 22-47: DAI_GBL_INT_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	GRP1_TRG_EN	Group 1 Trigger Enable. The DAI_GBL_INT_EN.GRP1_TRG_EN bit enables the trigger-out for group 1.
18 (R/W)	GRP0_TRG_EN	Group 0 Trigger Enable. The DAI_GBL_INT_EN.GRP0_TRG_EN bit enables the trigger-out for group 0.
17 (R/W)	GRP1_INT_EN	Group 1 Interrupt Enable. The DAI_GBL_INT_EN.GRP1_INT_EN bit enables the interrupt for group 1.
16 (R/W)	GRP0_INT_EN	Group 0 Interrupt Enable. The DAI_GBL_INT_EN.GRP0_INT_EN bit enables the interrupt for group 0.

Table 22-47: DAI_GBL_INT_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	GRP1_SP3BINT_EN	SP3B Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP3BINT_EN bit enables the interrupt or trigger-out for SPORT3 B when it is part of group 1.
14 (R/W)	GRP1_SP3AINT_EN	SP3A Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP3AINT_EN bit enables the interrupt or trigger-out for SPORT3 A when it is part of group 1.
13 (R/W)	GRP1_SP2BINT_EN	SP2B Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP2BINT_EN bit enables the interrupt or trigger-out for SPORT2 B when it is part of group 1.
12 (R/W)	GRP1_SP2AINT_EN	SP2A Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP2AINT_EN bit enables the interrupt or trigger-out for SPORT2 A when it is part of group 1.
11 (R/W)	GRP1_SP1BINT_EN	SP1B Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP1BINT_EN bit enables the interrupt or trigger-out for SPORT1 B when it is part of group 1.
10 (R/W)	GRP1_SP1AINT_EN	SP1A Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP1AINT_EN bit enables the interrupt or trigger-out for SPORT1 A when it is part of group 1.
9 (R/W)	GRP1_SP0BINT_EN	SP0B Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP0BINT_EN bit enables the interrupt or trigger-out for SPORT0 B when it is part of group 1.
8 (R/W)	GRP1_SP0AINT_EN	SP0A Interrupt/Trigger Group 1 Enable. The DAI_GBL_INT_EN.GRP1_SP0AINT_EN bit enables the interrupt or trigger-out for SPORT0 A when it is part of group 1.
7 (R/W)	GRP0_SP3BINT_EN	SP3B Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP3BINT_EN bit enables the interrupt or trigger-out for SPORT3 B when it is part of group 0.
6 (R/W)	GRP0_SP3AINT_EN	SP3A Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP3AINT_EN bit enables the interrupt or trigger-out for SPORT3 A when it is part of group 0.
5 (R/W)	GRP0_SP2BINT_EN	SP2B Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP2BINT_EN bit enables the interrupt or trigger-out for SPORT2 B when it is part of group 0.

Table 22-47: DAI_GBL_INT_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	GRP0_SP2AINT_EN	SP2A Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP2AINT_EN bit enables the interrupt or trigger-out for SPORT2 A when it is part of group 0.
3 (R/W)	GRP0_SP1BINT_EN	SP1B Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP1BINT_EN bit enables the interrupt or trigger-out for SPORT1 B when it is part of group 0.
2 (R/W)	GRP0_SP1AINT_EN	SP1A Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SP1AINT_EN bit enables the interrupt or trigger-out for SPORT1 A when it is part of group 0.
1 (R/W)	GRP0_SPOBINT_EN	SP0B Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SPOBINT_EN bit enables the interrupt or trigger-out for SPORT0 B when it is part of group 0.
0 (R/W)	GRP0_SPOAINT_EN	SP0A Interrupt/Trigger Group 0 Enable. The DAI_GBL_INT_EN.GRP0_SPOAINT_EN bit enables the interrupt or trigger-out for SPORT0 A when it is part of group 0.

Global SPORT Enable Register

The `DAI_GBL_SP_EN` register indicate the status (signal high or low) for each pin. The individual bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

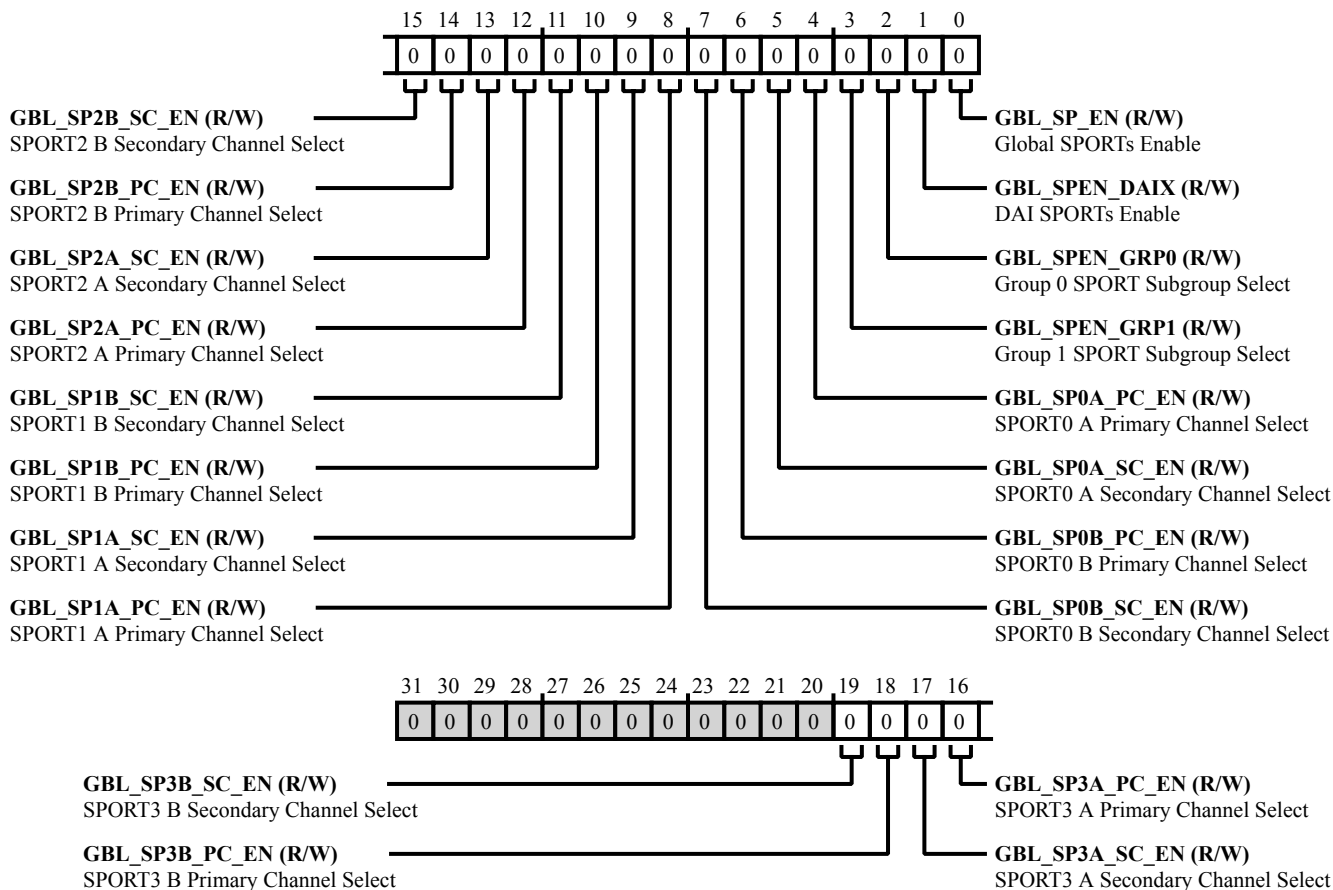


Figure 22-35: `DAI_GBL_SP_EN` Register Diagram

Table 22-48: `DAI_GBL_SP_EN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	<code>GBL_SP3B_SC_EN</code>	SPORT3 B Secondary Channel Select. The <code>DAI_GBL_SP_EN.GBL_SP3B_SC_EN</code> bit selects SPORT3 B as the secondary channel for global enable.
18 (R/W)	<code>GBL_SP3B_PC_EN</code>	SPORT3 B Primary Channel Select. The <code>DAI_GBL_SP_EN.GBL_SP3B_PC_EN</code> bit selects SPORT3 B as the primary channel for global enable.

Table 22-48: DAI_GBL_SP_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	GBL_SP3A_SC_EN	SPORT3 A Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP3A_SC_EN bit selects SPORT3 A as the secondary channel for global enable.
16 (R/W)	GBL_SP3A_PC_EN	SPORT3 A Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP3A_PC_EN bit selects SPORT3 A as the primary channel for global enable.
15 (R/W)	GBL_SP2B_SC_EN	SPORT2 B Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP2B_SC_EN bit selects SPORT2 B as the secondary channel for global enable.
14 (R/W)	GBL_SP2B_PC_EN	SPORT2 B Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP2B_PC_EN bit selects SPORT2 B as the primary channel for global enable.
13 (R/W)	GBL_SP2A_SC_EN	SPORT2 A Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP2A_SC_EN bit selects SPORT2 A as the secondary channel for global enable.
12 (R/W)	GBL_SP2A_PC_EN	SPORT2 A Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP2A_PC_EN bit selects SPORT2 A as the primary channel for global enable.
11 (R/W)	GBL_SP1B_SC_EN	SPORT1 B Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP1B_SC_EN bit selects SPORT1 B as the secondary channel for global enable.
10 (R/W)	GBL_SP1B_PC_EN	SPORT1 B Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP1B_PC_EN bit selects SPORT1 B as the primary channel for global enable.
9 (R/W)	GBL_SP1A_SC_EN	SPORT1 A Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP1A_SC_EN bit selects SPORT1 A as the secondary channel for global enable.
8 (R/W)	GBL_SP1A_PC_EN	SPORT1 A Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP1A_PC_EN bit selects SPORT1 A as the primary channel for global enable.
7 (R/W)	GBL_SP0B_SC_EN	SPORT0 B Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP0B_SC_EN bit selects SPORT0 B as the secondary channel for global enable.
6 (R/W)	GBL_SP0B_PC_EN	SPORT0 B Primary Channel Select. Selects SPORT0 B primary channel for global enable.

Table 22-48: DAI_GBL_SP_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	GBL_SP0A_SC_EN	SPORT0 A Secondary Channel Select. The DAI_GBL_SP_EN.GBL_SP0A_SC_EN bit selects SPORT0 A as the secondary channel for global enable.
4 (R/W)	GBL_SP0A_PC_EN	SPORT0 A Primary Channel Select. The DAI_GBL_SP_EN.GBL_SP0A_PC_EN bit selects SPORT0 A as the primary channel for global enable.
3 (R/W)	GBL_SPEN_GRP1	Group 1 SPORT Subgroup Select. The DAI_GBL_SP_EN.GBL_SPEN_GRP1 bit creates a subgroup of selected SPORTs.
2 (R/W)	GBL_SPEN_GRP0	Group 0 SPORT Subgroup Select. The DAI_GBL_SP_EN.GBL_SPEN_GRP0 bit creates a subgroup of selected SPORTs.
1 (R/W)	GBL_SPEN_DAIX	DAI SPORTs Enable. The DAI_GBL_SP_EN.GBL_SPEN_DAIX bit enables the SPORTs selected within a DAI.
0 (R/W)	GBL_SP_EN	Global SPORTs Enable. The DAI_GBL_SP_EN.GBL_SP_EN bit enables the selected SPORTs.

Falling-Edge Interrupt Mask Register

The `DAI_IMSK_FE` register masks and unmasks interrupts generated on the falling edge of a waveform. Note that any of the Group E signals can be mapped to any of the Miscellaneous interrupts (9-0).

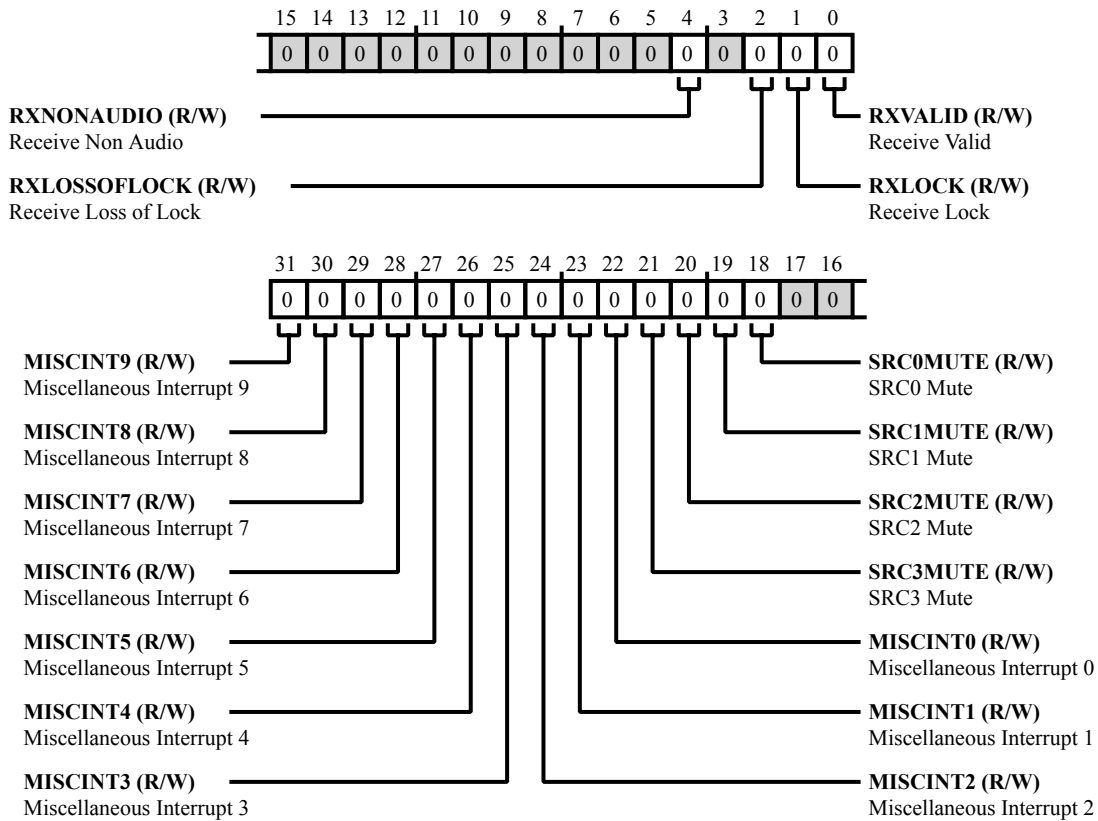


Figure 22-36: `DAI_IMSK_FE` Register Diagram

Table 22-49: `DAI_IMSK_FE` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. Setting the <code>DAI_IMSK_FE.MISCINT9</code> bit unmasks the interrupt for the falling edge of the routed signal.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. Setting the <code>DAI_IMSK_FE.MISCINT8</code> bit unmasks the interrupt for the falling edge of the routed signal.
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. Setting the <code>DAI_IMSK_FE.MISCINT7</code> bit unmasks the interrupt for the falling edge of the routed signal.

Table 22-49: DAI_IMSK_FE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. Setting the DAI_IMSK_FE.MISCINT6 bit unmask the interrupt for the falling edge of the routed signal.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. Setting the DAI_IMSK_FE.MISCINT5 bit unmask the interrupt for the falling edge of the routed signal.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. Setting the DAI_IMSK_FE.MISCINT4 bit unmask the interrupt for the falling edge of the routed signal.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. Setting the DAI_IMSK_FE.MISCINT3 bit unmask the interrupt for the falling edge of the routed signal.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. Setting the DAI_IMSK_FE.MISCINT2 bit unmask the interrupt for the falling edge of the routed signal.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. Setting the DAI_IMSK_FE.MISCINT1 bit unmask the interrupt for the falling edge of the routed signal.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. Setting the DAI_IMSK_FE.MISCINT0 bit unmask the interrupt for the falling edge of the routed signal.
21 (R/W)	SRC3MUTE	SRC3 Mute. The DAI_IMSK_FE.SRC3MUTE bit masks or unmask the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
20 (R/W)	SRC2MUTE	SRC2 Mute. The DAI_IMSK_FE.SRC2MUTE bit masks or unmask the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
19 (R/W)	SRC1MUTE	SRC1 Mute. The DAI_IMSK_FE.SRC1MUTE bit masks or unmask the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.

Table 22-49: DAI_IMSK_FE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SRC0MUTE	SRC0 Mute. The DAI_IMSK_FE.SRC0MUTE bit masks or unmasks the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
4 (R/W)	RXNONAUDIO	Receive Non Audio. The DAI_IMSK_FE.RXNONAUDIO bit masks or unmasks the non audio frame mode interrupt for the falling edge of this interrupt. If the channel status indicates non-PCM audio, the NONAUDIO bit flag is set.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The DAI_IMSK_FE.RXLOSSOFLOCK bit masks or unmasks the emphasis loss of lock interrupt for the falling edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver loses the lock of biphasic stream.
1 (R/W)	RXLOCK	Receive Lock. The DAI_IMSK_FE.RXLOCK bit masks or unmasks the S/PDIF receiver lock for the falling edge of this interrupt. This interrupt occurs when the S/PDIF receiver locks to the S/PDIF stream.
0 (R/W)	RXVALID	Receive Valid. Setting the DAI_IMSK_FE.RXVALID bit unmasks the interrupt for the falling edge of the routed signal. This interrupt is set based on whether data received by the S/PDIF is linear PCM or non-linear audio data.

Core Interrupt Priority Assignment Register

The `DAI_IMSK_PRI` register masks interrupts for DAI high or DAI low interrupt priority.

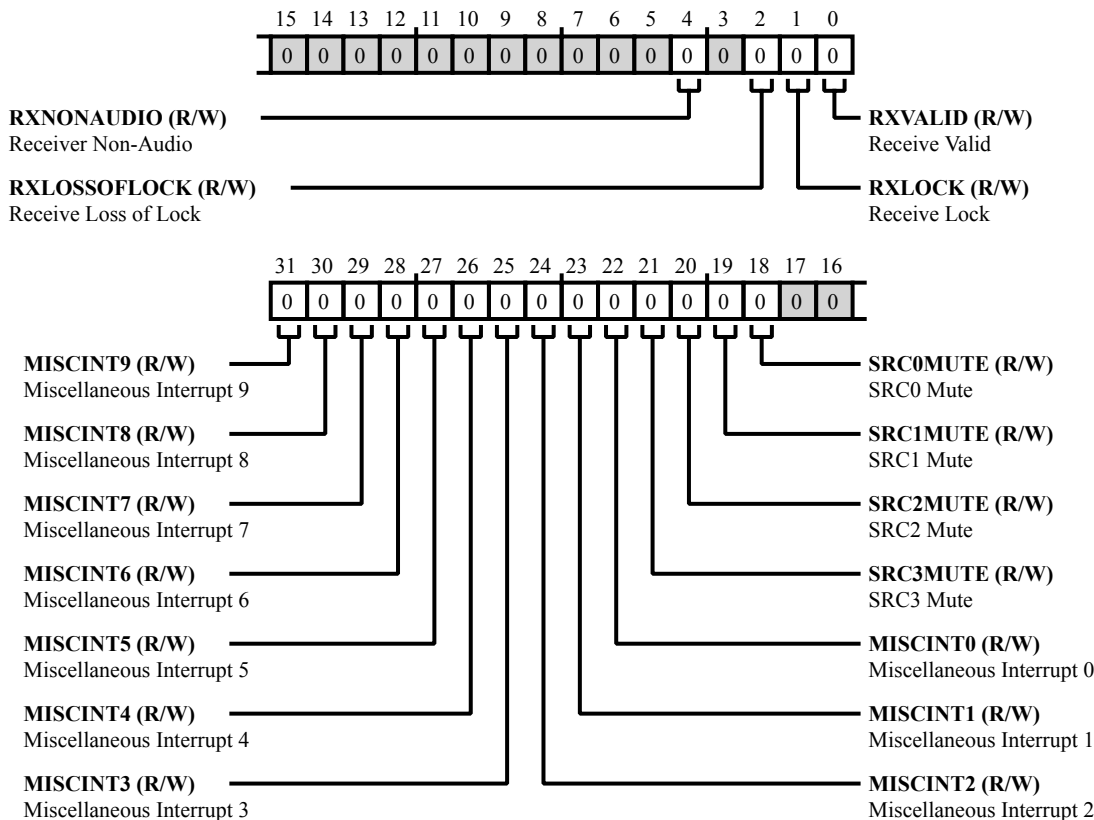


Figure 22-37: DAI_IMSK_PRI Register Diagram

Table 22-50: DAI_IMSK_PRI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. If the <code>DAI_IMSK_PRI.MISCINT9</code> bit is cleared (=0), the interrupt is mapped to <code>INTR_DAI_IRQL</code> signal from the SEC. If the <code>DAI_IMSK_PRI.MISCINT9</code> bit is set (=1) the interrupt is mapped to <code>INTR_DAI_IRQH</code> signal from the SEC.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. If the <code>DAI_IMSK_PRI.MISCINT8</code> bit is cleared (=0), the interrupt is mapped to <code>INTR_DAI_IRQL</code> signal from the SEC. If the <code>DAI_IMSK_PRI.MISCINT8</code> bit is set (=1) the interrupt is mapped to <code>INTR_DAI_IRQH</code> signal from the SEC.

Table 22-50: DAI_IMSK_PRI Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. If the DAI_IMSK_PRI.MISCINT7 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT7 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. If the DAI_IMSK_PRI.MISCINT6 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT6 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. If the DAI_IMSK_PRI.MISCINT5 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT5 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. If the DAI_IMSK_PRI.MISCINT4 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT4 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. If the DAI_IMSK_PRI.MISCINT3 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT3 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. If the DAI_IMSK_PRI.MISCINT2 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT2 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. If the DAI_IMSK_PRI.MISCINT1 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT1 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. If the DAI_IMSK_PRI.MISCINT0 bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.MISCINT0 bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
21 (R/W)	SRC3MUTE	SRC3 Mute. If the DAI_IMSK_PRI.SRC3MUTE bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI.SRC3MUTE bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.

Table 22-50: DAI_IMSK_PRI Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	SRC2MUTE	SRC2 Mute. If the DAI_IMSK_PRI . SRC2MUTE bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI . SRC2MUTE bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
19 (R/W)	SRC1MUTE	SRC1 Mute. If the DAI_IMSK_PRI . SRC1MUTE bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI . SRC1MUTE bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
18 (R/W)	SRC0MUTE	SRC0 Mute. If the DAI_IMSK_PRI . SRC0MUTE bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI . SRC0MUTE bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
4 (R/W)	RXNONAUDIO	Receiver Non-Audio. If the DAI_IMSK_PRI . RXNONAUDIO bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI . RXNONAUDIO bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The DAI_IMSK_PRI . RXLOSSOFLOCK bit masks or unmasks the emphasis loss of lock interrupt for the falling edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver loses the lock of biphase stream.
1 (R/W)	RXLOCK	Receive Lock. If the DAI_IMSK_PRI . RXLOCK bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the DAI_IMSK_PRI . RXLOCK bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
0 (R/W)	RXVALID	Receive Valid. If the DAI_IMSK_PRI . RXVALID bit is cleared (=0), the interrupt is mapped to the INTR_DAI_IRQL from the SEC. If the DAI_IMSK_PRI . RXVALID bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.

Rising-Edge Interrupt Mask Register

The `DAI_IMSK_RE` register masks and unmasks interrupts generated on the rising edge of a waveform. Note that any of the Group E signals can be mapped to any of the miscellaneous interrupts (9-0).

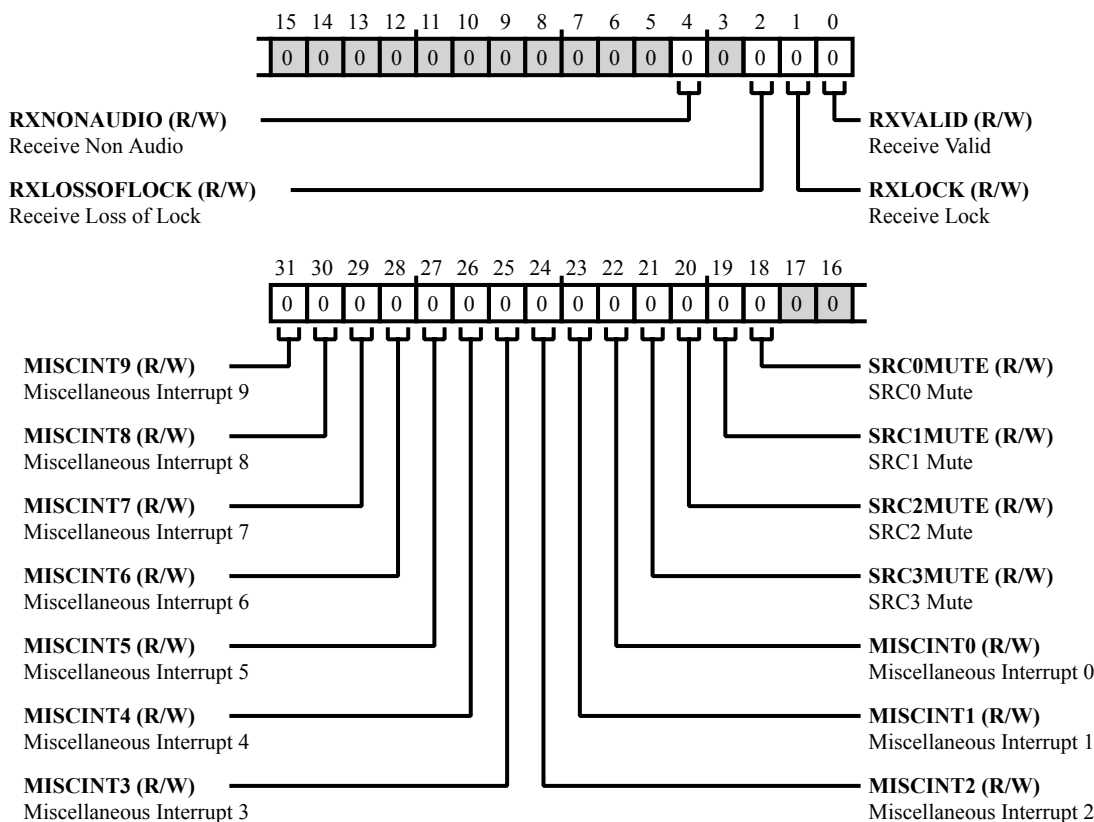


Figure 22-38: `DAI_IMSK_RE` Register Diagram

Table 22-51: `DAI_IMSK_RE` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. Setting the <code>DAI_IMSK_RE.MISCINT9</code> bit unmask the interrupt for the rising edge of the routed signal.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. Setting the <code>DAI_IMSK_RE.MISCINT8</code> bit unmask the interrupt for the rising edge of the routed signal.
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. Setting the <code>DAI_IMSK_RE.MISCINT7</code> bit unmask the interrupt for the rising edge of the routed signal.

Table 22-51: DAI_IMSK_RE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. Setting the DAI_IMSK_RE.MISCINT6 bit unmask the interrupt for the rising edge of the routed signal.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. Setting the DAI_IMSK_RE.MISCINT5 bit unmask the interrupt for the rising edge of the routed signal.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. Setting the DAI_IMSK_RE.MISCINT4 bit unmask the interrupt for the rising edge of the routed signal.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. Setting the DAI_IMSK_RE.MISCINT3 bit unmask the interrupt for the rising edge of the routed signal.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. Setting the DAI_IMSK_RE.MISCINT2 bit unmask the interrupt for the rising edge of the routed signal.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. Setting the DAI_IMSK_RE.MISCINT1 bit unmask the interrupt for the rising edge of the routed signal.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. Setting the DAI_IMSK_RE.MISCINT0 bit unmask the interrupt for the rising edge of the routed signal.
21 (R/W)	SRC3MUTE	SRC3 Mute. The DAI_IMSK_RE.SRC3MUTE bit masks or unmask the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
20 (R/W)	SRC2MUTE	SRC2 Mute. The DAI_IMSK_RE.SRC2MUTE bit masks or unmask the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
19 (R/W)	SRC1MUTE	SRC1 Mute. The DAI_IMSK_RE.SRC1MUTE bit masks or unmask the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.

Table 22-51: DAI_IMSK_RE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SRC0MUTE	SRC0 Mute. The <code>DAI_IMSK_RE.SRC0MUTE</code> bit masks or unmasks the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
4 (R/W)	RXNONAUDIO	Receive Non Audio. The <code>DAI_IMSK_RE.RXNONAUDIO</code> bit masks or unmasks the non audio frame mode interrupt for the rising edge of this interrupt. If the channel status indicates non-PCM audio, the NONAUDIO bit flag is set.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The <code>DAI_IMSK_RE.RXLOSSOFLOCK</code> bit masks or unmasks the emphasis loss of lock interrupt for the rising edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver loses the lock of biphase stream.
1 (R/W)	RXLOCK	Receive Lock. The <code>DAI_IMSK_RE.RXLOCK</code> bit masks or unmasks the S/PDIF receiver lock for the rising edge of this interrupt. This interrupt occurs when the S/PDIF receiver locks to the S/PDIF stream.
0 (R/W)	RXVALID	Receive Valid. Setting the <code>DAI_IMSK_RE.RXVALID</code> bit unmasks the interrupt for the rising edge of the routed signal. This interrupt is set based on whether data received by the S/PDIF is linear PCM or non-linear audio data.

High Priority Interrupt Latch Register

The `DAI_IRPTL_H` register holds the high priority latched interrupt status for interrupt requests that have been unmasked (enabled) by the `DAI_IMSK_FE`, `DAI_IMSK_RE` registers and mapped to the `INTR_DAI_IRQH` signal in the SEC by `DAI_IMSK_PRI` registers. If a bit in this register is already set and the corresponding interrupt is masked in the `DAI_IMSK_FE`, `DAI_IMSK_RE` or `DAI_IMSK_PRI` registers, the latch holds its old value, leaving the interrupt asserted until its mask registers (`DAI_IMSK_FE`, `DAI_IMSK_RE` or `DAI_IMSK_PRI`) are reset by software with a `W1C` operation.

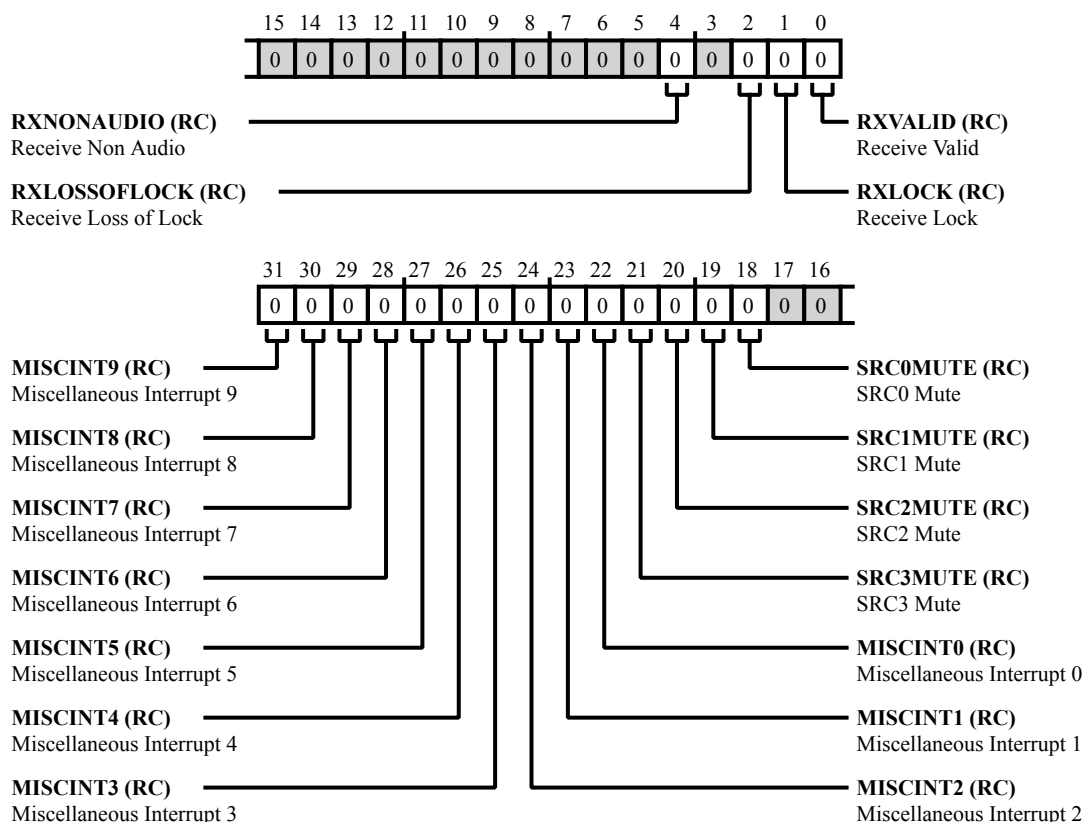


Figure 22-39: `DAI_IRPTL_H` Register Diagram

Table 22-52: `DAI_IRPTL_H` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (RC/NW)	MISCINT9	Miscellaneous Interrupt 9. The <code>DAI_IRPTL_H.MISCINT9</code> bit is set if the interrupt is mapped to the <code>INTR_DAI_IRQH</code> signal in the SEC using the <code>DAI_IMSK_PRI</code> register.
30 (RC/NW)	MISCINT8	Miscellaneous Interrupt 8. The <code>DAI_IRPTL_H.MISCINT8</code> bit is set if the interrupt is mapped to the <code>INTR_DAI_IRQH</code> signal in the SEC using the <code>DAI_IMSK_PRI</code> register.

Table 22-52: DAI_IRPTL_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (RC/NW)	MISCINT7	Miscellaneous Interrupt 7. The DAI_IRPTL_H.MISCINT7 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
28 (RC/NW)	MISCINT6	Miscellaneous Interrupt 6. The DAI_IRPTL_H.MISCINT6 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
27 (RC/NW)	MISCINT5	Miscellaneous Interrupt 5. The DAI_IRPTL_H.MISCINT5 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
26 (RC/NW)	MISCINT4	Miscellaneous Interrupt 4. The DAI_IRPTL_H.MISCINT4 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
25 (RC/NW)	MISCINT3	Miscellaneous Interrupt 3. The DAI_IRPTL_H.MISCINT3 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
24 (RC/NW)	MISCINT2	Miscellaneous Interrupt 2. The DAI_IRPTL_H.MISCINT2 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
23 (RC/NW)	MISCINT1	Miscellaneous Interrupt 1. The DAI_IRPTL_H.MISCINT1 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
22 (RC/NW)	MISCINT0	Miscellaneous Interrupt 0. The DAI_IRPTL_H.MISCINT0 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
21 (RC/NW)	SRC3MUTE	SRC3 Mute. The DAI_IRPTL_H.SRC3MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
20 (RC/NW)	SRC2MUTE	SRC2 Mute. The DAI_IRPTL_H.SRC2MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
19 (RC/NW)	SRC1MUTE	SRC1 Mute. The DAI_IRPTL_H.SRC1MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.

Table 22-52: DAI_IRPTL_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RC/NW)	SRC0MUTE	SRC0 Mute. The DAI_IRPTL_H.SRC0MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
4 (RC/NW)	RXNONAUDIO	Receive Non Audio. The DAI_IRPTL_H.RXNONAUDIO bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
2 (RC/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The DAI_IRPTL_H.RXLOSSOFLOCK bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
1 (RC/NW)	RXLOCK	Receive Lock. The DAI_IRPTL_H.RXLOCK bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
0 (RC/NW)	RXVALID	Receive Valid. The DAI_IRPTL_H.RXVALID bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.

Shadow High Priority Interrupt Latch Register

The `DAI_IRPTL_HS` register is the shadow register of the `DAI_IRPTL_H` register. Its content is the same as the `DAI_IRPTL_H` register. Reading the `DAI_IRPTL_HS` register does not affect its contents while reading the contents of the `DAI_IRPTL_H` registers clears it.

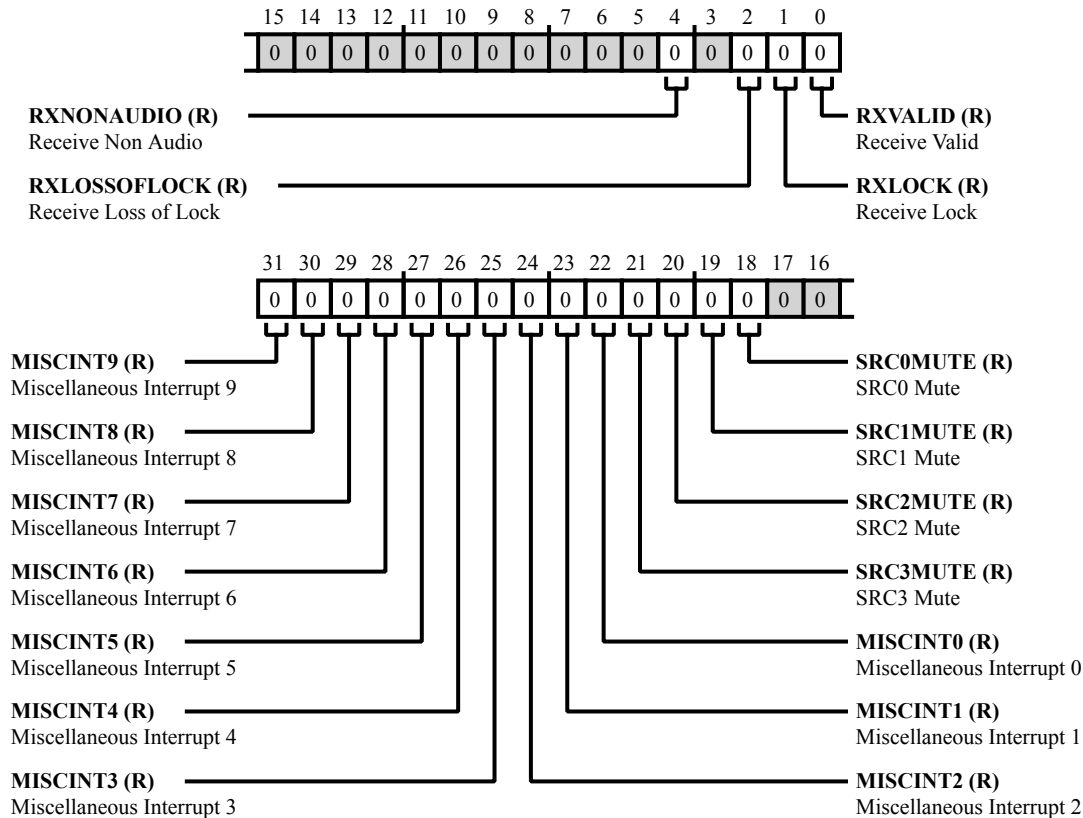


Figure 22-40: DAI_IRPTL_HS Register Diagram

Table 22-53: DAI_IRPTL_HS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MISCINT9	Miscellaneous Interrupt 9. The <code>DAI_IRPTL_HS.MISCINT9</code> bit is the shadow of the <code>DAI_IRPTL_H.MISCINT9</code> bit and contains the same content. Reading the <code>DAI_IRPTL_HS.MISCINT9</code> bit does not affect its contents while reading the <code>DAI_IRPTL_H.MISCINT9</code> bit clears it.
30 (R/NW)	MISCINT8	Miscellaneous Interrupt 8. The <code>DAI_IRPTL_HS.MISCINT8</code> bit is the shadow of the <code>DAI_IRPTL_H.MISCINT8</code> bit and contains the same content. Reading the <code>DAI_IRPTL_HS.MISCINT8</code> bit does not affect its contents while reading the <code>DAI_IRPTL_H.MISCINT8</code> bit clears it.

Table 22-53: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	MISCINT7	Miscellaneous Interrupt 7. The DAI_IRPTL_HS.MISCINT7 bit is the shadow of the DAI_IRPTL_H.MISCINT7 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT7 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT7 bit clears it.
28 (R/NW)	MISCINT6	Miscellaneous Interrupt 6. The DAI_IRPTL_HS.MISCINT6 bit is the shadow of the DAI_IRPTL_H.MISCINT6 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT6 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT6 bit clears it.
27 (R/NW)	MISCINT5	Miscellaneous Interrupt 5. The DAI_IRPTL_HS.MISCINT5 bit is the shadow of the DAI_IRPTL_H.MISCINT5 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT5 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT5 bit clears it.
26 (R/NW)	MISCINT4	Miscellaneous Interrupt 4. The DAI_IRPTL_HS.MISCINT4 bit is the shadow of the DAI_IRPTL_H.MISCINT4 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT4 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT4 bit clears it.
25 (R/NW)	MISCINT3	Miscellaneous Interrupt 3. The DAI_IRPTL_HS.MISCINT3 bit is the shadow of the DAI_IRPTL_H.MISCINT3 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT3 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT3 bit clears it.
24 (R/NW)	MISCINT2	Miscellaneous Interrupt 2. The DAI_IRPTL_HS.MISCINT2 bit is the shadow of the DAI_IRPTL_H.MISCINT2 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT2 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT2 bit clears it.
23 (R/NW)	MISCINT1	Miscellaneous Interrupt 1. The DAI_IRPTL_HS.MISCINT1 bit is the shadow of the DAI_IRPTL_H.MISCINT1 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT1 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT1 bit clears it.

Table 22-53: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/NW)	MISCINT0	Miscellaneous Interrupt 0. The DAI_IRPTL_HS.MISCINT0 bit is the shadow of the DAI_IRPTL_H.MISCINT0 bit and contains the same content. Reading the DAI_IRPTL_HS.MISCINT0 bit does not affect its contents while reading the DAI_IRPTL_H.MISCINT0 bit clears it.
21 (R/NW)	SRC3MUTE	SRC3 Mute. The DAI_IRPTL_HS.SRC3MUTE bit is the shadow of the DAI_IRPTL_H.SRC3MUTE bit and contains the same content. Reading the DAI_IRPTL_HS.SRC3MUTE bit does not affect its contents while reading the DAI_IRPTL_H.SRC3MUTE bit clears it.
20 (R/NW)	SRC2MUTE	SRC2 Mute. The DAI_IRPTL_HS.SRC2MUTE bit is the shadow of the DAI_IRPTL_H.SRC2MUTE bit and contains the same content. Reading the DAI_IRPTL_HS.SRC2MUTE bit does not affect its contents while reading the DAI_IRPTL_H.SRC2MUTE bit clears it.
19 (R/NW)	SRC1MUTE	SRC1 Mute. The DAI_IRPTL_HS.SRC1MUTE bit is the shadow of the DAI_IRPTL_H.SRC1MUTE bit and contains the same content. Reading the DAI_IRPTL_HS.SRC1MUTE bit does not affect its contents while reading the DAI_IRPTL_H.SRC1MUTE bit clears it.
18 (R/NW)	SRC0MUTE	SRC0 Mute. The DAI_IRPTL_HS.SRC0MUTE bit is the shadow of the DAI_IRPTL_H.SRC0MUTE bit and contains the same content. Reading the DAI_IRPTL_HS.SRC0MUTE bit does not affect its contents while reading the DAI_IRPTL_H.SRC0MUTE bit clears it.
4 (R/NW)	RXNONAUDIO	Receive Non Audio. The DAI_IRPTL_HS.RXNONAUDIO bit is the shadow of DAI_IRPTL_H.RXNONAUDIO bit and contains the same content. Reading this bit does not affect its content.
2 (R/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The DAI_IRPTL_HS.RXLOSSOFLOCK bit is the shadow of DAI_IRPTL_H.RXLOSSOFLOCK bit and contains the same content. Reading this bit does not affect its content.
1 (R/NW)	RXLOCK	Receive Lock. The DAI_IRPTL_HS.RXLOCK bit is the shadow of DAI_IRPTL_H.RXLOCK bit and contains the same content. Reading this bit does not affect its content.

Table 22-53: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	RXVALID	Receive Valid. Setting the DAI_IRPTL_HS.RXVALID bit is the shadow of DAI_IRPTL_H.RXVALID bit and contains the same content. Reading this bit does not affect its content.

Low Priority Interrupt Latch Register

The `DAI_IRPTL_L` register holds the low priority latched interrupt status for interrupt requests that have been unmasked (enabled) by the `DAI_IMSK_FE`, `DAI_IMSK_RE` or `DAI_IMSK_PRI` registers. If a bit in `DAI_IRPTL_L` is already set and the corresponding interrupt is masked in `DAI_IMSK_FE`, `DAI_IMSK_RE` or `DAI_IMSK_PRI` registers, the latch holds its old value, leaving the interrupt asserted until its masked registers (`DAI_IMSK_FE`, `DAI_IMSK_RE` or `DAI_IMSK_PRI`) are reset by software with a W1C operation.

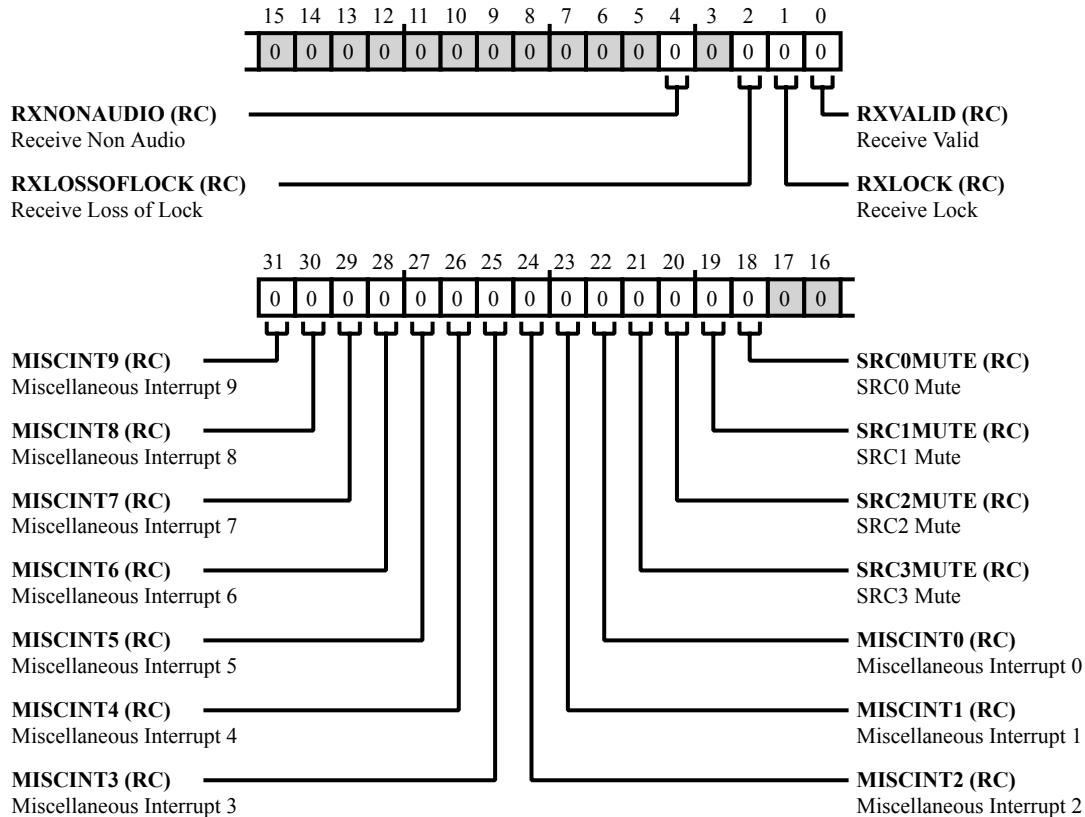


Figure 22-41: `DAI_IRPTL_L` Register Diagram

Table 22-54: `DAI_IRPTL_L` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (RC/NW)	MISCINT9	Miscellaneous Interrupt 9. The <code>DAI_IRPTL_L.MISCINT9</code> bit is set if the interrupt is mapped to the <code>INTR_DAI_IRQL</code> signal in the SEC using the <code>DAI_IMSK_PRI</code> register.
30 (RC/NW)	MISCINT8	Miscellaneous Interrupt 8. The <code>DAI_IRPTL_L.MISCINT8</code> bit is set if the interrupt is mapped to the <code>INTR_DAI_IRQL</code> signal in the SEC using the <code>DAI_IMSK_PRI</code> register.

Table 22-54: DAI_IRPTL_L Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (RC/NW)	MISCINT7	Miscellaneous Interrupt 7. The DAI_IRPTL_L.MISCINT7 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
28 (RC/NW)	MISCINT6	Miscellaneous Interrupt 6. The DAI_IRPTL_L.MISCINT6 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
27 (RC/NW)	MISCINT5	Miscellaneous Interrupt 5. The DAI_IRPTL_L.MISCINT5 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
26 (RC/NW)	MISCINT4	Miscellaneous Interrupt 4. The DAI_IRPTL_L.MISCINT4 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
25 (RC/NW)	MISCINT3	Miscellaneous Interrupt 3. The DAI_IRPTL_L.MISCINT3 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
24 (RC/NW)	MISCINT2	Miscellaneous Interrupt 2. The DAI_IRPTL_L.MISCINT2 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
23 (RC/NW)	MISCINT1	Miscellaneous Interrupt 1. The DAI_IRPTL_L.MISCINT1 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
22 (RC/NW)	MISCINT0	Miscellaneous Interrupt 0. The DAI_IRPTL_L.MISCINT0 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
21 (RC/NW)	SRC3MUTE	SRC3 Mute. The DAI_IRPTL_L.SRC3MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
20 (RC/NW)	SRC2MUTE	SRC2 Mute. The DAI_IRPTL_L.SRC2MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
19 (RC/NW)	SRC1MUTE	SRC1 Mute. The DAI_IRPTL_L.SRC1MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.

Table 22-54: DAI_IRPTL_L Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RC/NW)	SRC0MUTE	SRC0 Mute. The DAI_IRPTL_L.SRC0MUTE bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
4 (RC/NW)	RXNONAUDIO	Receive Non Audio. The DAI_IRPTL_L.RXNONAUDIO bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
2 (RC/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The DAI_IRPTL_L.RXLOSSOFLOCK bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
1 (RC/NW)	RXLOCK	Receive Lock. The DAI_IRPTL_L.RXLOCK bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
0 (RC/NW)	RXVALID	Receive Valid. The DAI_IRPTL_L.RXVALID bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.

Shadow Low Priority Interrupt Latch Register

The `DAI_IRPTL_LS` register is the shadow register of the `DAI_IRPTL_L` register. Its content is the same as the `DAI_IRPTL_L` register. Reading the `DAI_IRPTL_LS` register does not affect its contents while reading the contents of the `DAI_IRPTL_L` registers clears it.

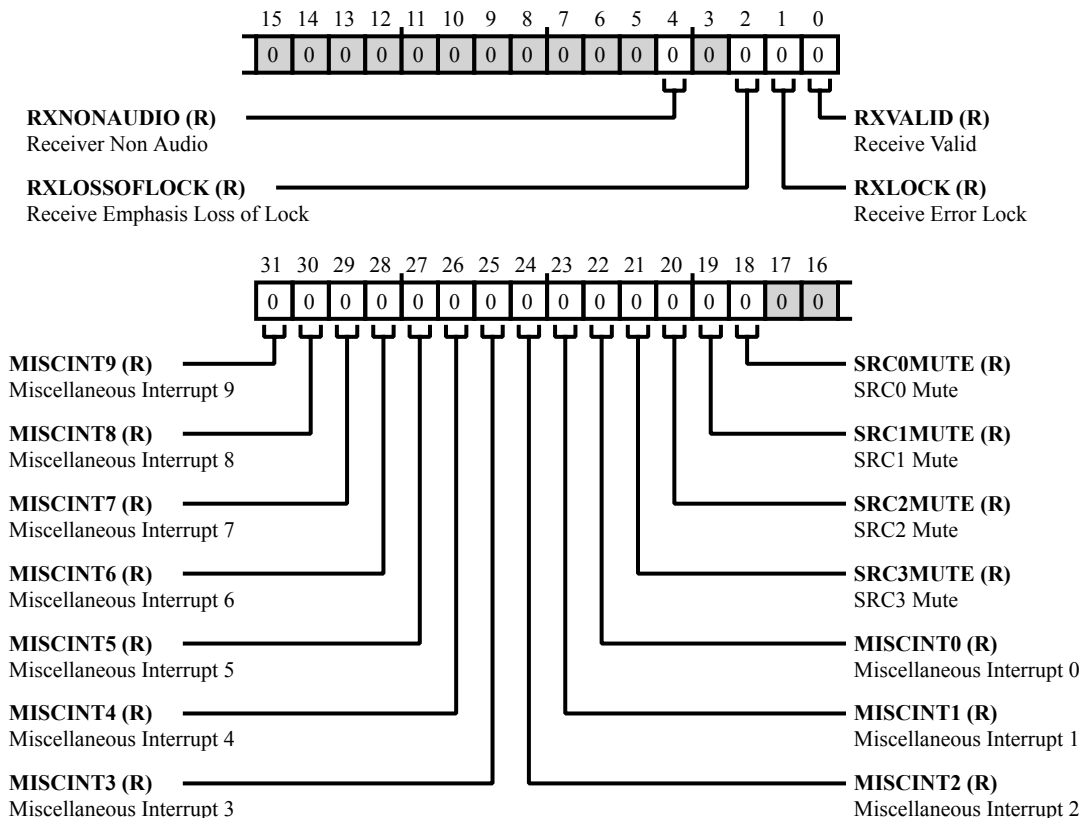


Figure 22-42: `DAI_IRPTL_LS` Register Diagram

Table 22-55: `DAI_IRPTL_LS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MISCINT9	Miscellaneous Interrupt 9. The <code>DAI_IRPTL_LS.MISCINT9</code> bit is the shadow of the <code>DAI_IRPTL_L.MISCINT9</code> bit and contains the same content. Reading the <code>DAI_IRPTL_LS.MISCINT9</code> bit does not affect its contents while reading the <code>DAI_IRPTL_L.MISCINT9</code> bit clears it.
30 (R/NW)	MISCINT8	Miscellaneous Interrupt 8. The <code>DAI_IRPTL_LS.MISCINT8</code> bit is the shadow of the <code>DAI_IRPTL_L.MISCINT8</code> bit and contains the same content. Reading the <code>DAI_IRPTL_LS.MISCINT8</code> bit does not affect its contents while reading the <code>DAI_IRPTL_L.MISCINT8</code> bit clears it.

Table 22-55: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	MISCINT7	Miscellaneous Interrupt 7. The DAI_IRPTL_LS.MISCINT7 bit is the shadow of the DAI_IRPTL_L.MISCINT7 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT7 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT7 bit clears it.
28 (R/NW)	MISCINT6	Miscellaneous Interrupt 6. The DAI_IRPTL_LS.MISCINT6 bit is the shadow of the DAI_IRPTL_L.MISCINT6 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT6 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT6 bit clears it.
27 (R/NW)	MISCINT5	Miscellaneous Interrupt 5. The DAI_IRPTL_LS.MISCINT5 bit is the shadow of the DAI_IRPTL_L.MISCINT5 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT5 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT5 bit clears it.
26 (R/NW)	MISCINT4	Miscellaneous Interrupt 4. The DAI_IRPTL_LS.MISCINT4 bit is the shadow of the DAI_IRPTL_L.MISCINT4 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT4 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT4 bit clears it.
25 (R/NW)	MISCINT3	Miscellaneous Interrupt 3. The DAI_IRPTL_LS.MISCINT3 bit is the shadow of the DAI_IRPTL_L.MISCINT3 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT3 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT3 bit clears it.
24 (R/NW)	MISCINT2	Miscellaneous Interrupt 2. The DAI_IRPTL_LS.MISCINT2 bit is the shadow of the DAI_IRPTL_L.MISCINT2 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT2 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT2 bit clears it.
23 (R/NW)	MISCINT1	Miscellaneous Interrupt 1. The DAI_IRPTL_LS.MISCINT1 bit is the shadow of the DAI_IRPTL_L.MISCINT1 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT1 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT1 bit clears it.

Table 22-55: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/NW)	MISCINT0	Miscellaneous Interrupt 0. The DAI_IRPTL_LS.MISCINT0 bit is the shadow of the DAI_IRPTL_L.MISCINT0 bit and contains the same content. Reading the DAI_IRPTL_LS.MISCINT0 bit does not affect its contents while reading the DAI_IRPTL_L.MISCINT0 bit clears it.
21 (R/NW)	SRC3MUTE	SRC3 Mute. The DAI_IRPTL_LS.SRC3MUTE bit is the shadow of the DAI_IRPTL_L.SRC3MUTE bit and contains the same content. Reading the DAI_IRPTL_LS.SRC3MUTE bit does not affect its contents while reading the DAI_IRPTL_L.SRC3MUTE bit clears it.
20 (R/NW)	SRC2MUTE	SRC2 Mute. The DAI_IRPTL_LS.SRC2MUTE bit is the shadow of the DAI_IRPTL_L.SRC2MUTE bit and contains the same content. Reading the DAI_IRPTL_LS.SRC2MUTE bit does not affect its contents while reading the DAI_IRPTL_L.SRC2MUTE bit clears it.
19 (R/NW)	SRC1MUTE	SRC1 Mute. The DAI_IRPTL_LS.SRC1MUTE bit is the shadow of the DAI_IRPTL_L.SRC1MUTE bit and contains the same content. Reading the DAI_IRPTL_LS.SRC1MUTE bit does not affect its contents while reading the DAI_IRPTL_L.SRC1MUTE bit clears it.
18 (R/NW)	SRC0MUTE	SRC0 Mute. The DAI_IRPTL_LS.SRC0MUTE bit is the shadow of the DAI_IRPTL_L.SRC0MUTE bit and contains the same content. Reading the DAI_IRPTL_LS.SRC0MUTE bit does not affect its contents while reading the DAI_IRPTL_L.SRC0MUTE bit clears it.
4 (R/NW)	RXNONAUDIO	Receiver Non Audio. The DAI_IRPTL_LS.RXNONAUDIO bit is the shadow of the DAI_IRPTL_L.RXNONAUDIO bit and contains the same content. Reading the DAI_IRPTL_LS.RXNONAUDIO bit does not affect its contents while reading the DAI_IRPTL_L.RXNONAUDIO bit clears it.
2 (R/NW)	RXLOSSOFLOCK	Receive Emphasis Loss of Lock. The DAI_IRPTL_LS.RXLOSSOFLOCK bit is the shadow of DAI_IRPTL_L.RXLOSSOFLOCK bit and contains the same content. Reading this bit does not affect its content. Reading the DAI_IRPTL_L.RXLOSSOFLOCK bit clears it.
1 (R/NW)	RXLOCK	Receive Error Lock. The DAI_IRPTL_LS.RXLOCK bit is the shadow of DAI_IRPTL_L.RXLOCK bit and contains the same content. Reading this bit does not affect its content. Reading the DAI_IRPTL_L.RXLOCK bit clears it.

Table 22-55: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	RXVALID	<p>Receive Valid.</p> <p>The <code>DAI_IRPTL_LS.RXVALID</code> bit is the shadow of <code>DAI_IRPTL_L.RXVALID</code> bit and contains the same content. Reading this bit does not affect its content. Reading the <code>DAI_IRPTL_L.RXVALID</code> bit clears it.</p>

Miscellaneous Control Register 0

The `DAI_MISC0` register allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion. This register belongs to group E which routes control signals and provides a means of connecting signals between groups.

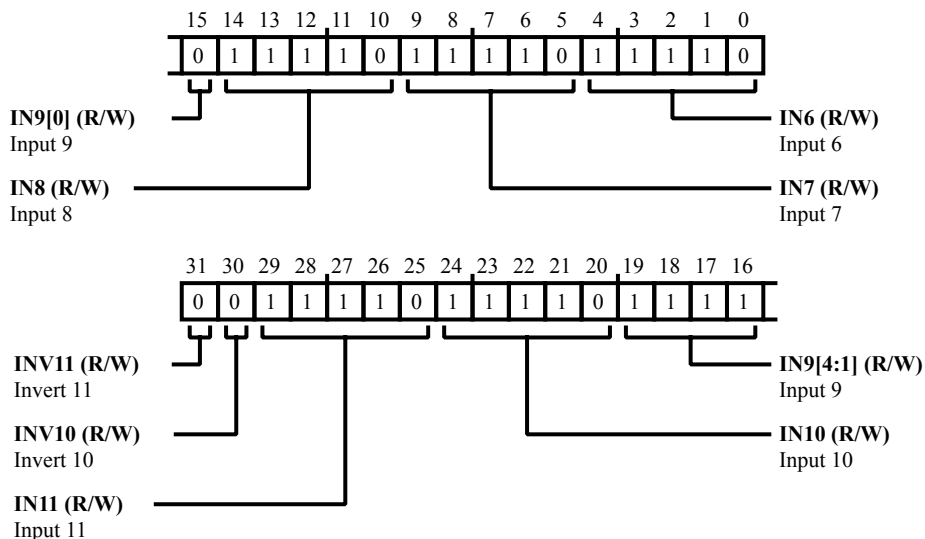


Figure 22-43: DAI_MISC0 Register Diagram

Table 22-56: DAI_MISC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	INV11	Invert 11. The <code>DAI_MISC0.INV11</code> bit field inverts miscellaneous A5 input.
30 (R/W)	INV10	Invert 10. The <code>DAI_MISC0.INV10</code> bit field inverts miscellaneous A4 input.
29:25 (R/W)	IN11	Input 11. The <code>DAI_MISC0.IN11</code> bit field configures miscellaneous A5 input.
24:20 (R/W)	IN10	Input 10. The <code>DAI_MISC0.IN10</code> bit field configures miscellaneous A4 input.
19:15 (R/W)	IN9	Input 9. The <code>DAI_MISC0.IN9</code> bit field configures miscellaneous A3 input/miscellaneous Interrupt 9(DAI interrupt 31).
14:10 (R/W)	IN8	Input 8. The <code>DAI_MISC0.IN8</code> bit field configures miscellaneous A2 input/miscellaneous Interrupt 8(DAI interrupt 30).

Table 22-56: DAI_MISC0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN7	Input 7. The <code>DAI_MISC0.IN7</code> bit field configures miscellaneous A1 input/miscellaneous Interrupt 7(DAI interrupt 29).
4:0 (R/W)	IN6	Input 6. The <code>DAI_MISC0.IN6</code> bit field configures miscellaneous A0 input/miscellaneous Interrupt 6(DAI interrupt 28).

Miscellaneous Control Register 1

The `DAI_MISC1` register allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion. This register belongs to group E which routes control signals and provides a means of connecting signals between groups.

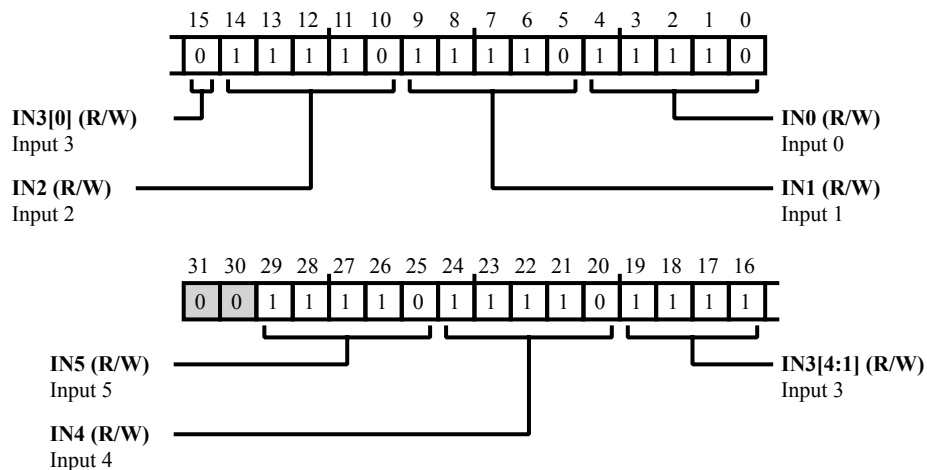


Figure 22-44: DAI_MISC1 Register Diagram

Table 22-57: DAI_MISC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input 5. The <code>DAI_MISC1 . IN5</code> bit field configures miscellaneous Interrupt 5(DAI interrupt 27).
24:20 (R/W)	IN4	Input 4. The <code>DAI_MISC1 . IN4</code> bit field configures miscellaneous Interrupt 4(DAI interrupt 26).
19:15 (R/W)	IN3	Input 3. The <code>DAI_MISC1 . IN3</code> bit field configures miscellaneous Interrupt 3(DAI interrupt 25).
14:10 (R/W)	IN2	Input 2. The <code>DAI_MISC1 . IN2</code> bit field configures miscellaneous Interrupt 2(DAI interrupt 24).
9:5 (R/W)	IN1	Input 1. The <code>DAI_MISC1 . IN1</code> bit field configures miscellaneous Interrupt 1(DAI interrupt 23).

Table 22-57: DAI_MISC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input 0. The <code>DAI_MISC1.IN0</code> bit field configures miscellaneous Interrupt 0(DAI interrupt 22).

Pin Buffer Enable Register 0

The `DAI_PBEN0` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

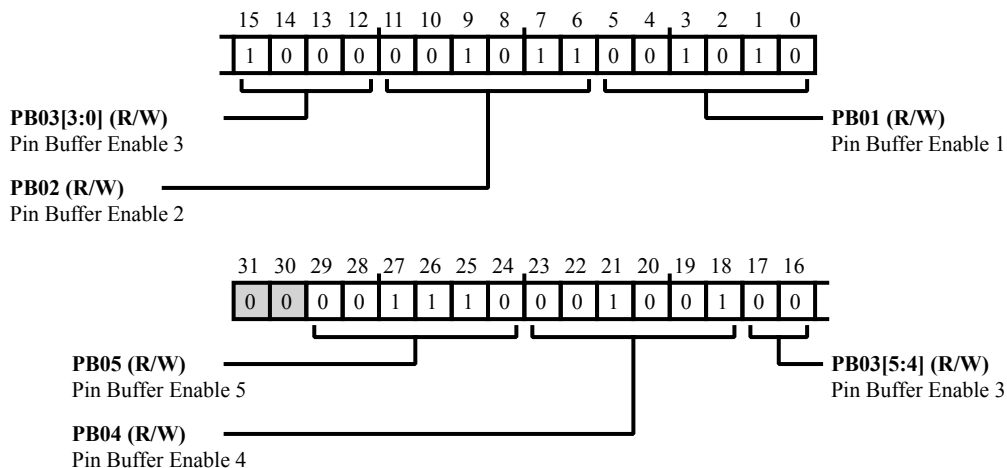


Figure 22-45: DAI_PBEN0 Register Diagram

Table 22-58: DAI_PBEN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB05	Pin Buffer Enable 5. The <code>DAI_PBEN0.PB05</code> bit field is the pin buffer enable for DAI port 5.
23:18 (R/W)	PB04	Pin Buffer Enable 4. The <code>DAI_PBEN0.PB04</code> bit field is the pin buffer enable for DAI port 4.
17:12 (R/W)	PB03	Pin Buffer Enable 3. The <code>DAI_PBEN0.PB03</code> bit field is the pin buffer enable for DAI port 3.
11:6 (R/W)	PB02	Pin Buffer Enable 2. The <code>DAI_PBEN0.PB02</code> bit field is the pin buffer enable for DAI port 2.
5:0 (R/W)	PB01	Pin Buffer Enable 1. The <code>DAI_PBEN0.PB01</code> bit field is the pin buffer enable for DAI port 1.

Pin Buffer Enable Register 1

The `DAI_PBEN1` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

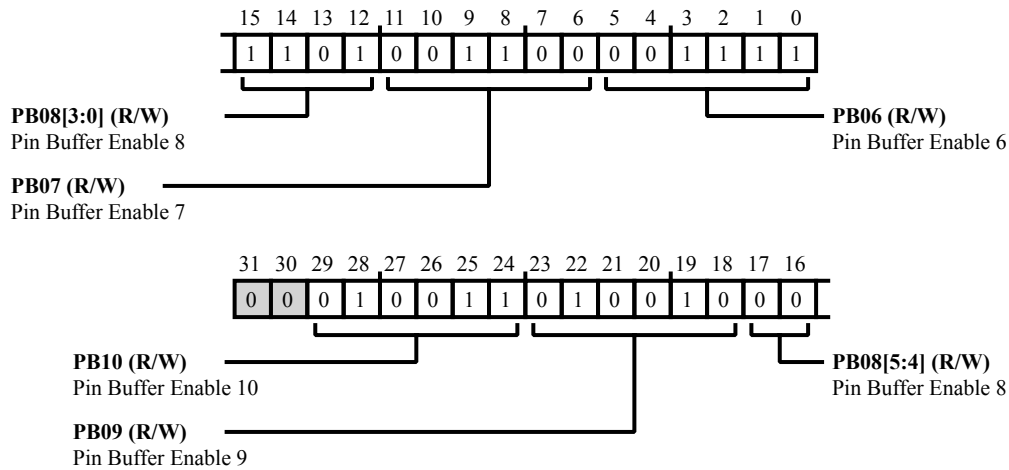


Figure 22-46: DAI_PBEN1 Register Diagram

Table 22-59: DAI_PBEN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB10	Pin Buffer Enable 10. The <code>DAI_PBEN1.PB10</code> bit field is the pin buffer enable for DAI port 10.
23:18 (R/W)	PB09	Pin Buffer Enable 9. The <code>DAI_PBEN1.PB09</code> bit field is the pin buffer enable for DAI port 9.
17:12 (R/W)	PB08	Pin Buffer Enable 8. The <code>DAI_PBEN1.PB08</code> bit field is the pin buffer enable for DAI port 8.
11:6 (R/W)	PB07	Pin Buffer Enable 7. The <code>DAI_PBEN1.PB07</code> bit field is the pin buffer enable for DAI port 7.
5:0 (R/W)	PB06	Pin Buffer Enable 6. The <code>DAI_PBEN1.PB06</code> bit field is the pin buffer enable for DAI port 6.

Pin Buffer Enable Register 2

The `DAI_PBEN2` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

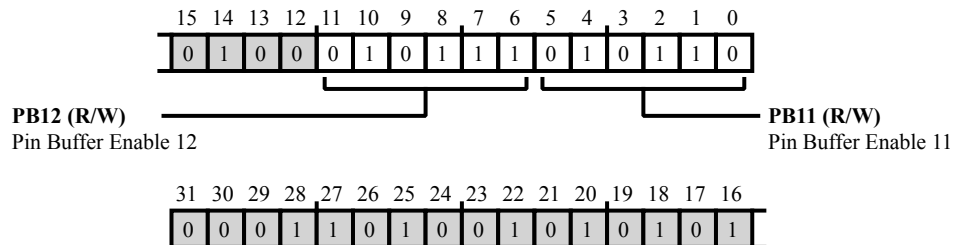


Figure 22-47: DAI_PBEN2 Register Diagram

Table 22-60: DAI_PBEN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:6 (R/W)	PB12	Pin Buffer Enable 12. The <code>DAI_PBEN2.PB12</code> bit field is the pin buffer enable for DAI port 12.
5:0 (R/W)	PB11	Pin Buffer Enable 11. The <code>DAI_PBEN2.PB11</code> bit field is the pin buffer enable for DAI port 11.

Pin Buffer Enable Register 3

The `DAI_PBEN3` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

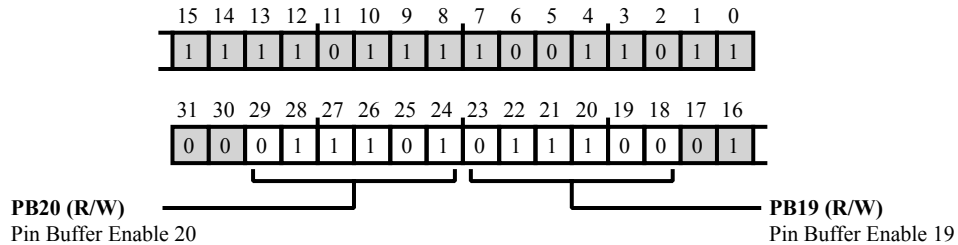


Figure 22-48: DAI_PBEN3 Register Diagram

Table 22-61: DAI_PBEN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB20	Pin Buffer Enable 20. The <code>DAI_PBEN3.PB20</code> bit field is the pin buffer enable for DAI port 20.
23:18 (R/W)	PB19	Pin Buffer Enable 19. The <code>DAI_PBEN3.PB19</code> bit field is the pin buffer enable for DAI port 19.

Pin Buffer Assignment Register 0

The `DAI_PIN0` register routes physical pins that are connected to a bonded pad.

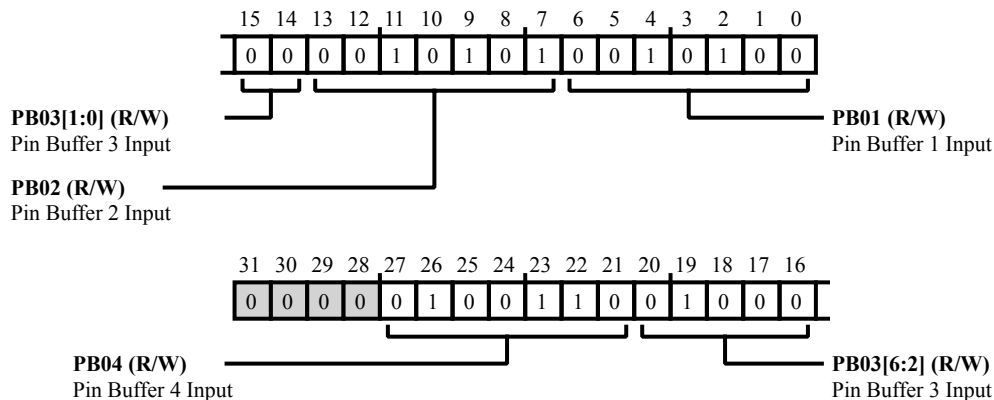


Figure 22-49: `DAI_PIN0` Register Diagram

Table 22-62: `DAI_PIN0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB04	Pin Buffer 4 Input. <code>DAI_PIN0.PB04</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN0.PB04</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB03	Pin Buffer 3 Input. <code>DAI_PIN0.PB03</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN0.PB03</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB02	Pin Buffer 2 Input. <code>DAI_PIN0.PB02</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN0.PB02</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB01	Pin Buffer 1 Input. <code>DAI_PIN0.PB01</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN0.PB01</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 1

The `DAI_PIN1` register routes physical pins that are connected to a bonded pad.

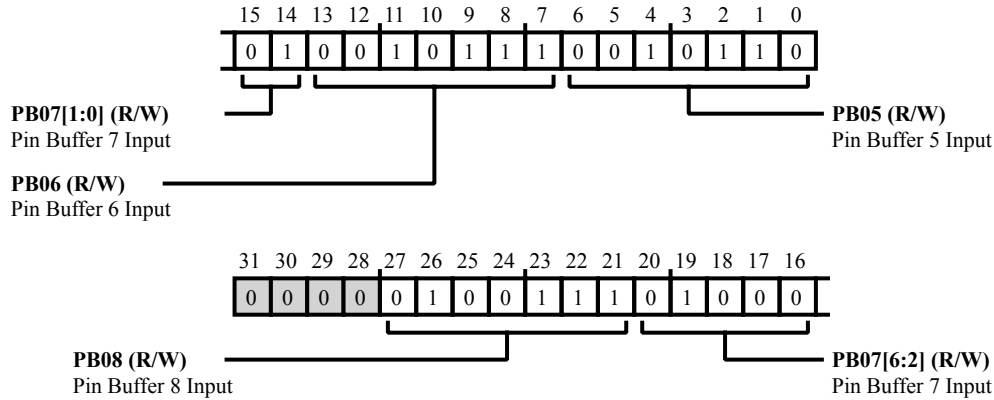


Figure 22-50: DAI_PIN1 Register Diagram

Table 22-63: DAI_PIN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB08	Pin Buffer 8 Input. DAI_PIN1 . PB08 holds the Source signal assignment that will be routed to the DAI_PIN1 . PB08 Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB07	Pin Buffer 7 Input. DAI_PIN1 . PB07 holds the Source signal assignment that will be routed to the DAI_PIN1 . PB07 Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB06	Pin Buffer 6 Input. DAI_PIN1 . PB06 holds the Source signal assignment that will be routed to the DAI_PIN1 . PB06 Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB05	Pin Buffer 5 Input. DAI_PIN1 . PB05 holds the Source signal assignment that will be routed to the DAI_PIN1 . PB05 Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 2

The `DAI_PIN2` register routes physical pins that are connected to a bonded pad.

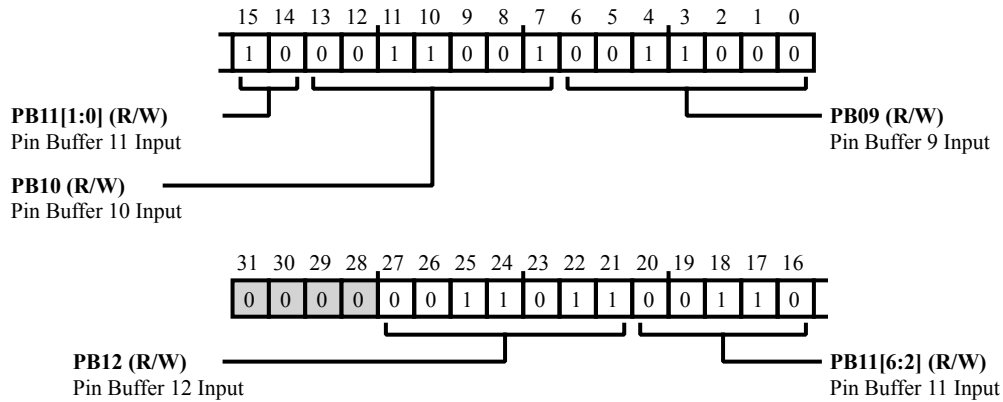


Figure 22-51: `DAI_PIN2` Register Diagram

Table 22-64: `DAI_PIN2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB12	Pin Buffer 12 Input. <code>DAI_PIN2.PB12</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN2.PB12</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB11	Pin Buffer 11 Input. <code>DAI_PIN2.PB11</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN2.PB11</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB10	Pin Buffer 10 Input. <code>DAI_PIN2.PB10</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN2.PB10</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB09	Pin Buffer 9 Input. <code>DAI_PIN2.PB09</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN2.PB09</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 4

The `DAI_PIN4` register routes physical pins that are connected to a bonded pad.

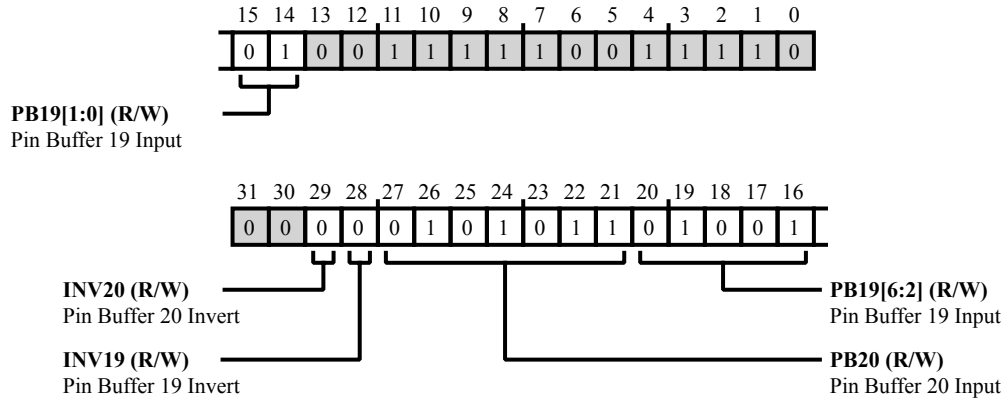


Figure 22-52: DAI_PIN4 Register Diagram

Table 22-65: DAI_PIN4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	INV20	Pin Buffer 20 Invert. <code>DAI_PIN4</code> . <code>INV20</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN4</code> . <code>INV20</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
28 (R/W)	INV19	Pin Buffer 19 Invert. <code>DAI_PIN4</code> . <code>INV19</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN4</code> . <code>INV19</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
27:21 (R/W)	PB20	Pin Buffer 20 Input. <code>DAI_PIN4</code> . <code>PB20</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN4</code> . <code>PB20</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB19	Pin Buffer 19 Input. <code>DAI_PIN4</code> . <code>PB19</code> holds the Source signal assignment that will be routed to the <code>DAI_PIN4</code> . <code>PB19</code> Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Status Register

The `DAI_PIN_STAT` register bits indicate the status (signal high or low) for each pin. The individual bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

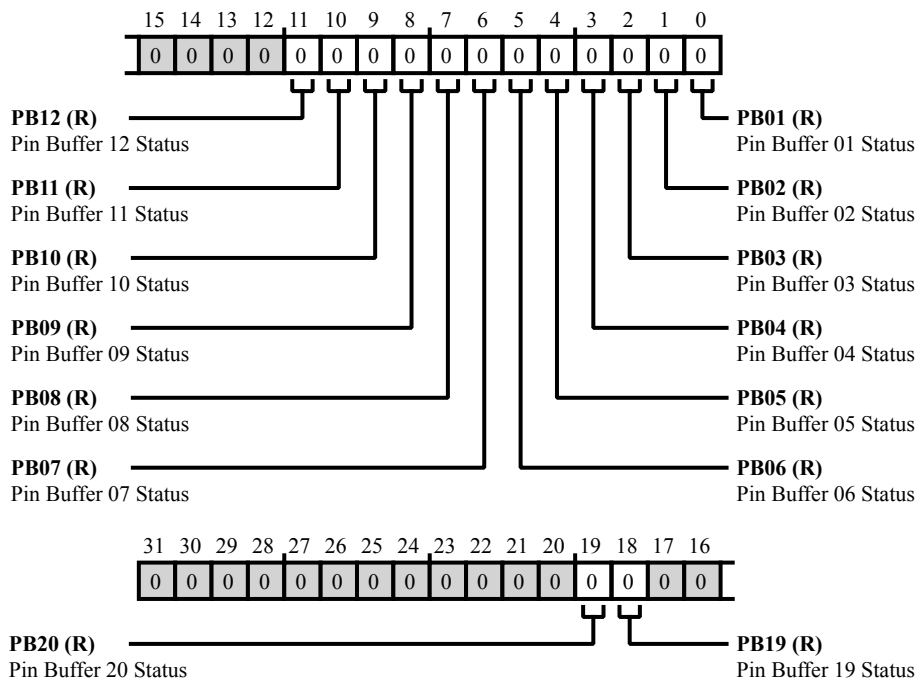


Figure 22-53: `DAI_PIN_STAT` Register Diagram

Table 22-66: `DAI_PIN_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/NW)	PB20	Pin Buffer 20 Status. The <code>DAI_PIN_STAT.PB20</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
18 (R/NW)	PB19	Pin Buffer 19 Status. The <code>DAI_PIN_STAT.PB19</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
11 (R/NW)	PB12	Pin Buffer 12 Status. The <code>DAI_PIN_STAT.PB12</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
10 (R/NW)	PB11	Pin Buffer 11 Status. The <code>DAI_PIN_STAT.PB11</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

Table 22-66: DAI_PIN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	PB10	Pin Buffer 10 Status. The <code>DAI_PIN_STAT.PB10</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
8 (R/NW)	PB09	Pin Buffer 09 Status. The <code>DAI_PIN_STAT.PB09</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
7 (R/NW)	PB08	Pin Buffer 08 Status. The <code>DAI_PIN_STAT.PB08</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
6 (R/NW)	PB07	Pin Buffer 07 Status. The <code>DAI_PIN_STAT.PB07</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
5 (R/NW)	PB06	Pin Buffer 06 Status. The <code>DAI_PIN_STAT.PB06</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
4 (R/NW)	PB05	Pin Buffer 05 Status. The <code>DAI_PIN_STAT.PB05</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
3 (R/NW)	PB04	Pin Buffer 04 Status. The <code>DAI_PIN_STAT.PB04</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
2 (R/NW)	PB03	Pin Buffer 03 Status. The <code>DAI_PIN_STAT.PB03</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
1 (R/NW)	PB02	Pin Buffer 02 Status. The <code>DAI_PIN_STAT.PB02</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
0 (R/NW)	PB01	Pin Buffer 01 Status. The <code>DAI_PIN_STAT.PB01</code> bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

23 Serial Port (SPORT)

The programmable serial ports (SPORTs) support various protocols for serial data communication and provide a glueless hardware interface to many industry-standard data converters and codecs. They have high data rates and dual half-duplex datapaths and are ideal for establishing a direct serial connection among two or more processors in a multiprocessor system, as many processors provide compatible serial interfaces.

The SPORT top module consists of two half SPORTs with identical functionality and programming requirements. Each half SPORT can be independently configured as either a transmitter or receiver and can be coupled with the other half SPORT within the same SPORT top module. Further, each half SPORT provides two synchronous half-duplex data lines to double the total supported throughput. As such, a single SPORT top module can be used to provide up to four unidirectional or up to two full-duplex data streams. Further channels are possible as well, but utilization of multiple SPORT top modules is required, thus requiring external connections to provide a common time base.

Features

An individual SPORT top module consists of two independently configurable SPORT halves with identical functionality. These SPORT halves offer the following features:

- Up to two bidirectional data lines - each half SPORT supports up to two transmit or receive channels, thus allowing two unidirectional streams into or out of each half SPORT and providing greater flexibility for serial communications. If full-duplex functionality is desired, two SPORT halves can be combined to enable dual-stream bidirectional communication.
- Six operating modes:
 1. Standard DSP serial mode
 2. I²S mode
 3. Left-Justified mode
 4. Right-Justified mode
 5. Multichannel (TDM) mode
 6. Packed mode

- Supports internally or externally generated clock.
- Support for both even and odd SCLK to SPORT clock (SPORT_CLK) ratios. If both data lines of a half SPORT are active, the maximum throughput is 2 x SPORT_CLK bps.
- Configurable rising or falling edge of the SPORT_CLK for driving and sampling data and frame syncs.
- Gated clock mode support for internally or externally generated clocks in DSP serial mode and stereo modes (left-justified and I²S mode).
- Supports frameless operation.
- Supports internally or externally generated frame sync signals.
- Programmable frame sync polarity.
- Programmable frame sync timing (synchronous to data or 1 SPORT clock in advance of it).
- Detection of prematurely received external frame syncs (with optional interrupt request generation).
- Programmable level-/edge-sensitivity for external frame syncs.
- Programmable (4–32-bit) data length, either in most significant bit (MSB) first or in least significant bit (LSB) first format, with optional sign-extension on received data.
- Optional 16-bit to 32-bit word packing (as receiver) and 32-bit to 16-bit word unpacking (as transmitter).
- Support for A-law and μ -law compression/decompression hardware companding, according to the G.711 specification, on transmitted/received words in all operating modes.
- Transmit underrun and receive overflow detection (with optional interrupt request generation).
- TDM mode transfers data on 128 contiguous channels from a stream of up to 1024 total channels (useful for H.100/H.110 and other telephony interfaces as a network communication scheme for multiple processors).
- Performs interrupt-driven, single word core transfers to and from on-chip or off-chip memory.
- Dedicated DMA channel for each SPORT half supporting autobuffer (for a repeated, identical range of transfers) and numerous descriptor-based (individual or repeated ranges of transfers with differing DMA parameters) modes.
- Master and slave trigger functionality.
- Unique transfer finish interrupt (TFI) signaling when the last transmit word is fully out of the transmit shift register.
- Multiplexer to internally connect critical timing signals between SPORT halves.
- Support for Global Sport Enable functionality, where enabling multiple SPORTs need to be synchronized so that all of them start and/or end at the same time.

Signal Descriptions

Each half SPORT module has five dedicated signals, as described in the *SPORT Signal Descriptions* table. The actual pin name varies with different SPORT halves. Individual SPORT halves do not share any of its signals across the pair that comprises the SPORT top module; however, it is possible to connect the clock and frame sync signals between the SPORT half pair, as explained in the [Multiplexer Logic](#) section.

All of the SPORT signals are multiplexed on the PORT pins, possibly sharing functionality with other peripherals on the device. By default, the PORT pins are in GPIO mode and must be reconfigured for SPORT functionality by setting the appropriate bits in the [PORT_FER](#) and [PORT_MUX](#) registers. Consult the processor datasheet for details regarding which ports the SPORT signals are available on, and be sure to configure the [PORT_MUX](#) register before the [PORT_FER](#) register.

Table 23-1: SPORT Signal Descriptions

Internal Node	Direction	Description
SPORT _x _CLK	I/O	Transmit or receive serial clock. Data and frame syncs are driven or sampled on this clock's edges. This signal can be either internally or externally generated.
SPORT _x _FS	I/O	Transmit or receive frame sync. The frame sync pulse initiates shifting of serial data. This signal is either internally or externally generated.
SPORT _x _D0	I/O	Primary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORT _x _D1	I/O	Secondary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORT _x _TDV	O	Multichannel transmit data valid. This signal is only active in multichannel transmit mode and is asserted during enabled slots, as defined by the channel selection registers (SPORT_CS0_A through SPORT_CS3_B).

The data channel signals are transmit signals when the serial port is configured in transmit mode ([SPORT_CTL_A.SPTRAN](#) = 1). They are receive signals when the serial port is configured in receive mode ([SPORT_CTL_A.SPTRAN](#) = 0). The following sections further describe the SPORT signals.

NOTE: These sections explicitly refer to the registers associated with half SPORT A, but the same concepts also apply to half SPORT B.

Serial Clock

The serial port clock ([SPORT_ACLK](#)) is either a receive serial clock or a transmit serial clock, depending on the transfer direction ([SPORT_CTL_A.SPTRAN](#)), governing when the data bits are serially shifted into or out of the SPORT and when the frame sync signal is driven (in internal frame sync mode) or sampled (in external frame sync mode). It can be internally generated from the processor's system clock ([SCLK0_0](#)) or externally provided. If the half SPORT is configured in internal clock mode ([SPORT_CTL_A.ICLK](#) = 1), then the [SPORT_DIV_A.CLKDIV](#) field specifies the divisor applied to [SCLK0_0](#) to generate the SPORT clock. As it is a

16-bit divisor, a wide range of serial clock rates is possible. Use the following equation to calculate the serial clock frequency:

$$\text{SPORT_ACLK} = [\text{SCLK0_0} \div (\text{SPORT_DIV_A.CLKDIV} + 1)]$$

From this, the following equation can be used to determine the value of `SPORT_DIV_A.CLKDIV`, given the `SCLK0_0` frequency and the desired frequency of the SPORT clock:

$$\text{SPORT_DIV_A.CLKDIV} = [(\text{SCLK0_0} \div \text{SPORT_ACLK}) - 1]$$

The half SPORT also supports a 1:1 `SPORT_ACLK` to `SCLK0_0` ratio (per the equations above, program the clock divisor field to zero). In this case, the resulting SPORT clock frequency is equal to `SCLK0_0`.

NOTE: Be careful not to exceed the maximum `SPORT_ACLK` frequency specified in the processor datasheet.

In certain operating modes, the SPORT can be configured to generate a gated clock, which is active only during valid data. In some applications, a SPORT uses it to generate a general-purpose clock in the system. In this case, enable the SPORT with the appropriate `SPORT_DIV_A.CLKDIV` divisor field in internal clock mode.

If a SPORT is configured in external clock mode (`SPORT_CTL_A.ICLK = 0`), the serial clock is an input signal, thus making the SPORT operate in slave mode. In this mode, the `SPORT_DIV_A.CLKDIV` is irrelevant and is ignored. The optional loopback capability provided by the internal SPORT multiplexer (SPMUX) block allows the slave SPORT to use the serial clock from the neighboring SPORT half in the same SPORT top module.

An externally-supplied serial clock does not need to be synchronous with the processor clocks. Further, the external clock can be a gated clock, but it must comply with the requirements described in the [Gated Clock Mode](#) section.

Refer to the product datasheet for exact AC timing specifications.

Frame Sync

The SPORT frame sync (`SPORT_AFS`) signal is either a receive frame sync or a transmit frame sync, depending on the transfer direction (`SPORT_CTL_A.SPTRAN`), which is used to determine the start of a new word or frame. When this signal goes active, the serial port starts serially shifting data into or out of the SPORT. The frame sync signal can be internally generated based on its serial clock (`SPORT_ACLK`) or externally provided, as configured by the `SPORT_CTL_A.IFS` bit.

If the half SPORT is configured to generate frame syncs (`SPORT_CTL_A.IFS = 1`), then the `SPORT_DIV_A.FSDIV` field specifies the divisor used to generate the periodic `SPORT_AFS` signal from the SPORT clock. As this is a 16-bit divisor, a wide range of frame sync rates to initiate periodic transfers is possible. Whether the serial clock is internally or externally generated, this divisor is a count of SPORT clock cycles between frame sync pulses, the formula for which is:

$$\text{Number of serial clocks between frame syncs} = (\text{SPORT_DIV_A.FSDIV} + 1)$$

From this, the following equation can be used to determine the value of `SPORT_DIV_A.FSDIV`, given the serial clock frequency and the desired frame sync frequency:

$$\text{SPORT_DIV_A.FSDIV} = [(\text{SPORT_ACLK} \div \text{SPORT_AFS}) - 1]$$

The frame sync is continuously active when `SPORT_DIV_A.FSDIV=0`. The value of `SPORT_DIV_A.FSDIV` cannot be less than the serial word length minus one (the value of the `SPORT_CTL_A.SLEN` bit field). Failure to adhere to this guideline can cause an external device to abort the current operation or cause other unpredictable results.

NOTE: After enabling the SPORT, the first internal frame sync appears after a delay of `SPORT_DIV_A.FSDIV + 3` SPORT clock cycles.

If a SPORT is configured for external frame syncs (`SPORT_CTL_A.IFS = 0`), then `SPORT_AFS` is an input signal and the `SPORT_DIV_A.FSDIV` field of the `SPORT_DIV_A` register is irrelevant and ignored. By default, this external signal is level-sensitive, but it can be configured as an edge-sensitive signal by setting the `SPORT_CTL_A.FSED` bit. The frame sync is expected to be synchronous with the serial clock. If not, it must meet the timing requirements that appear in the datasheet.

The SPORT can be used as a counter for dividing an external clock to generate periodic pulses or periodic interrupts. To do so, enable the SPORT with the appropriate `SPORT_DIV_A.FSDIV` divisor field with the SPORT configured for an external clock and internal data-independent frame syncs.

In some of the operating modes, the SPORT can be programmed to treat the frame sync signal as an optional signal by clearing the `SPORT_CTL_A.FSR` bit. Even with this bit cleared, the SPORT requires a single frame sync assertion to start the continuous transfers, after which it is ignored (for externally supplied frame syncs) or not generated (for internally-generated frame syncs). Characteristics of the frame sync depend on the settings in the SPORT control registers and the operating mode of the SPORT. For more information, refer to the `SPORT_CTL_A` register.

Data Signals

Each half SPORT has two bidirectional data lines known as the primary (`SPORT_AD0`) and secondary (`SPORT_AD1`) data channels. Both of the data lines can be configured as either transmitters or receivers using the `SPORT_CTL_A.SPTRAN` bit, thus permitting dual unidirectional data streams to increase the data throughput of the SPORT.

NOTE: Configuring one transmit data channel and one receive data channel on a single half SPORT is not supported.

The primary and secondary data lines can be individually enabled or disabled using the `SPORT_CTL_A.SPENPRI` and the `SPORT_CTL_A.SPENSEC` bits, respectively. However, if using both, enable or disable them concurrently. These data lines operate in a synchronous manner (sharing a clock and frame sync) but have separate datapaths with unique data buffers, shift registers and optional companding logic. All of the SPORT control settings are common for both channels, but the single DMA channel per half SPORT serves both the primary and secondary data channels.

When a SPORT is configured in multichannel transmit mode, the data pins three-state during inactive channel slots, thus allowing multiple transmitters to operate on the same bus with different active channels.

See the [Architectural Concepts](#) section for more details about data transfer operation.

Transmit Data Valid Signal

The transmit data valid (`SPORT_ATDV`) signal is available only in transmit multichannel modes (including packed mode). It is driven active during enabled multichannel slots, and it is driven inactive during the disabled channels. In other words, the `SPORT_ATDV` signal is active when data is being driven to the data pins and inactive when the data pins are being three-stated. As such, the `SPORT_ATDV` signal can serve as an output-enable signal for the data transmit pin.

SRU Programming

The SPORT uses the SRU (signal routing unit) to connect the SPORT data, serial clock, frame sync, and external sync (if external synchronization is required). Inputs also must be routed through the SRU. Program the corresponding SRU registers to connect the outputs to the required destinations. For details of the routing, see [DAI Routing Capabilities](#) in the *Digital Audio Interface (DAI)* chapter.

SRU SPORT Receive Master

If the SPORT is operating as receive master, it must feed its master output clock back to its input clock. This is required to trigger the SPORT's state machine. Using SPORT 0A as an example receive master, programs should route `SPT0_ACLK_O` to `SPT0_ACLK_I`. This is not required if the SPORT is operating as a transmitter in master mode.

Functional Description

The following sections provide general information about the functionality of the processor's serial ports:

- [Architectural Concepts](#)
- [Data Types and Companding](#)
- [Transmit Path](#)
- [Receive Path](#)

ADSP-2156x SPORT Register List

The Serial Port (SPORT) controller, with its range of clock and frame synchronization options, supports a variety of serial communication protocols and provides a glueless hardware interface to many industry-standard data converters and CODECs. Each SPORT has two independent halves (A and B), and each half contains two channels (primary and secondary). A set of registers governs SPORT operations. For more information on SPORT functionality, see the SPORT register descriptions.

Table 23-2: ADSP-2156x SPORT Register List

Name	Description
<code>SPORT_CS0_A</code>	Half SPORT 'A' Multichannel 0-31 Select Register

Table 23-2: ADSP-2156x SPORT Register List (Continued)

Name	Description
SPORT_CS0_B	Half SPORT 'B' Multichannel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multichannel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multichannel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multichannel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multichannel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multichannel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

ADSP-2156x SPORT Interrupt List

Table 23-3: ADSP-2156x SPORT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
37	SPORT0_A_DMA	SPORT0 Channel A DMA	Level	0
38	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
39	SPORT0_B_DMA	SPORT0 Channel B DMA	Level	1
40	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
41	SPORT1_A_DMA	SPORT1 Channel A DMA	Level	2
42	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
43	SPORT1_B_DMA	SPORT1 Channel B DMA	Level	3
44	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
45	SPORT2_A_DMA	SPORT2 Channel A DMA	Level	4
46	SPORT2_A_STAT	SPORT2 Channel A Status	Level	
47	SPORT2_B_DMA	SPORT2 Channel B DMA	Level	5
48	SPORT2_B_STAT	SPORT2 Channel B Status	Level	
49	SPORT3_A_DMA	SPORT3 Channel A DMA	Level	6
50	SPORT3_A_STAT	SPORT3 Channel A Status	Level	
51	SPORT3_B_DMA	SPORT3 Channel B DMA	Level	7
52	SPORT3_B_STAT	SPORT3 Channel B Status	Level	
53	SPORT4_A_DMA	SPORT4 Channel A DMA	Level	10
54	SPORT4_A_STAT	SPORT4 Channel A Status	Level	
55	SPORT4_B_DMA	SPORT4 Channel B DMA	Level	11
56	SPORT4_B_STAT	SPORT4 Channel B Status	Level	
57	SPORT5_A_DMA	SPORT5 Channel A DMA	Level	12
58	SPORT5_A_STAT	SPORT5 Channel A Status	Level	
59	SPORT5_B_DMA	SPORT5 Channel B DMA	Level	13
60	SPORT5_B_STAT	SPORT5 Channel B Status	Level	
61	SPORT6_A_DMA	SPORT6 Channel A DMA	Level	14
62	SPORT6_A_STAT	SPORT6 Channel A Status	Level	
63	SPORT6_B_DMA	SPORT6 Channel B DMA	Level	15
64	SPORT6_B_STAT	SPORT6 Channel B Status	Level	
65	SPORT7_A_DMA	SPORT7 Channel A DMA	Level	16
66	SPORT7_A_STAT	SPORT7 Channel A Status	Level	
67	SPORT7_B_DMA	SPORT7 Channel B DMA	Level	17

Table 23-3: ADSP-2156x SPORT Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
68	SPORT7_B_STAT	SPORT7 Channel B Status	Level	
140	SPORT0_A_DMA_ERR	SPORT0 Channel A DMA Error	Level	
141	SPORT0_B_DMA_ERR	SPORT0 Channel B DMA Error	Level	
142	SPORT1_A_DMA_ERR	SPORT1 Channel A DMA Error	Level	
143	SPORT1_B_DMA_ERR	SPORT1 Channel B DMA Error	Level	
144	SPORT2_A_DMA_ERR	SPORT2 Channel A DMA Error	Level	
145	SPORT2_B_DMA_ERR	SPORT2 Channel B DMA Error	Level	
146	SPORT3_A_DMA_ERR	SPORT3 Channel A DMA Error	Level	
147	SPORT3_B_DMA_ERR	SPORT3 Channel B DMA Error	Level	
148	SPORT4_A_DMA_ERR	SPORT4 Channel A DMA Error	Level	
149	SPORT4_B_DMA_ERR	SPORT4 Channel B DMA Error	Level	
150	SPORT5_A_DMA_ERR	SPORT5 Channel A DMA Error	Level	
151	SPORT5_B_DMA_ERR	SPORT5 Channel B DMA Error	Level	
152	SPORT6_A_DMA_ERR	SPORT6 Channel A DMA Error	Level	
153	SPORT6_B_DMA_ERR	SPORT6 Channel B DMA Error	Level	
154	SPORT7_A_DMA_ERR	SPORT7 Channel A DMA Error	Level	
155	SPORT7_B_DMA_ERR	SPORT7 Channel B DMA Error	Level	

ADSP-2156x SPORT Trigger List

Table 23-4: ADSP-2156x SPORT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
17	SPORT0_A_DMA	SPORT0 Channel A DMA	Edge
18	SPORT0_B_DMA	SPORT0 Channel B DMA	Edge
19	SPORT1_A_DMA	SPORT1 Channel A DMA	Edge
20	SPORT1_B_DMA	SPORT1 Channel B DMA	Edge
21	SPORT2_A_DMA	SPORT2 Channel A DMA	Edge
22	SPORT2_B_DMA	SPORT2 Channel B DMA	Edge
23	SPORT3_A_DMA	SPORT3 Channel A DMA	Edge
24	SPORT3_B_DMA	SPORT3 Channel B DMA	Edge
25	SPORT4_A_DMA	SPORT4 Channel A DMA	Edge

Table 23-4: ADSP-2156x SPORT Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
26	SPORT4_B_DMA	SPORT4 Channel B DMA	Edge
27	SPORT5_A_DMA	SPORT5 Channel A DMA	Edge
28	SPORT5_B_DMA	SPORT5 Channel B DMA	Edge
29	SPORT6_A_DMA	SPORT6 Channel A DMA	Edge
30	SPORT6_B_DMA	SPORT6 Channel B DMA	Edge
31	SPORT7_A_DMA	SPORT7 Channel A DMA	Edge
32	SPORT7_B_DMA	SPORT7 Channel B DMA	Edge

Table 23-5: ADSP-2156x SPORT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
20	SPORT0_A_DMA	SPORT0 Channel A DMA	Pulse
21	SPORT0_B_DMA	SPORT0 Channel B DMA	Pulse
22	SPORT1_A_DMA	SPORT1 Channel A DMA	Pulse
23	SPORT1_B_DMA	SPORT1 Channel B DMA	Pulse
24	SPORT2_A_DMA	SPORT2 Channel A DMA	Pulse
25	SPORT2_B_DMA	SPORT2 Channel B DMA	Pulse
26	SPORT3_A_DMA	SPORT3 Channel A DMA	Pulse
27	SPORT3_B_DMA	SPORT3 Channel B DMA	Pulse
28	SPORT4_A_DMA	SPORT4 Channel A DMA	Pulse
29	SPORT4_B_DMA	SPORT4 Channel B DMA	Pulse
30	SPORT5_A_DMA	SPORT5 Channel A DMA	Pulse
31	SPORT5_B_DMA	SPORT5 Channel B DMA	Pulse
32	SPORT6_A_DMA	SPORT6 Channel A DMA	Pulse
33	SPORT6_B_DMA	SPORT6 Channel B DMA	Pulse
34	SPORT7_A_DMA	SPORT7 Channel A DMA	Pulse
35	SPORT7_B_DMA	SPORT7 Channel B DMA	Pulse

ADSP-2156x SPORT DMA Channel List

Table 23-6: ADSP-2156x SPORT DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0 Channel A DMA

Table 23-6: ADSP-2156x SPORT DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA1	SPORT0_B_DMA	SPORT0 Channel B DMA
DMA2	SPORT1_A_DMA	SPORT1 Channel A DMA
DMA3	SPORT1_B_DMA	SPORT1 Channel B DMA
DMA4	SPORT2_A_DMA	SPORT2 Channel A DMA
DMA5	SPORT2_B_DMA	SPORT2 Channel B DMA
DMA6	SPORT3_A_DMA	SPORT3 Channel A DMA
DMA7	SPORT3_B_DMA	SPORT3 Channel B DMA
DMA10	SPORT4_A_DMA	SPORT4 Channel A DMA
DMA11	SPORT4_B_DMA	SPORT4 Channel B DMA
DMA12	SPORT5_A_DMA	SPORT5 Channel A DMA
DMA13	SPORT5_B_DMA	SPORT5 Channel B DMA
DMA14	SPORT6_A_DMA	SPORT6 Channel A DMA
DMA15	SPORT6_B_DMA	SPORT6 Channel B DMA
DMA16	SPORT7_A_DMA	SPORT7 Channel A DMA
DMA17	SPORT7_B_DMA	SPORT7 Channel B DMA

Block Diagram

Each SPORT top module consists of two separate blocks, known as half SPORTs (HSPORT) A and B, each with identical functionality and programming models. The *Half Serial Port Block Diagram* shows a detailed block diagram of a half SPORT.

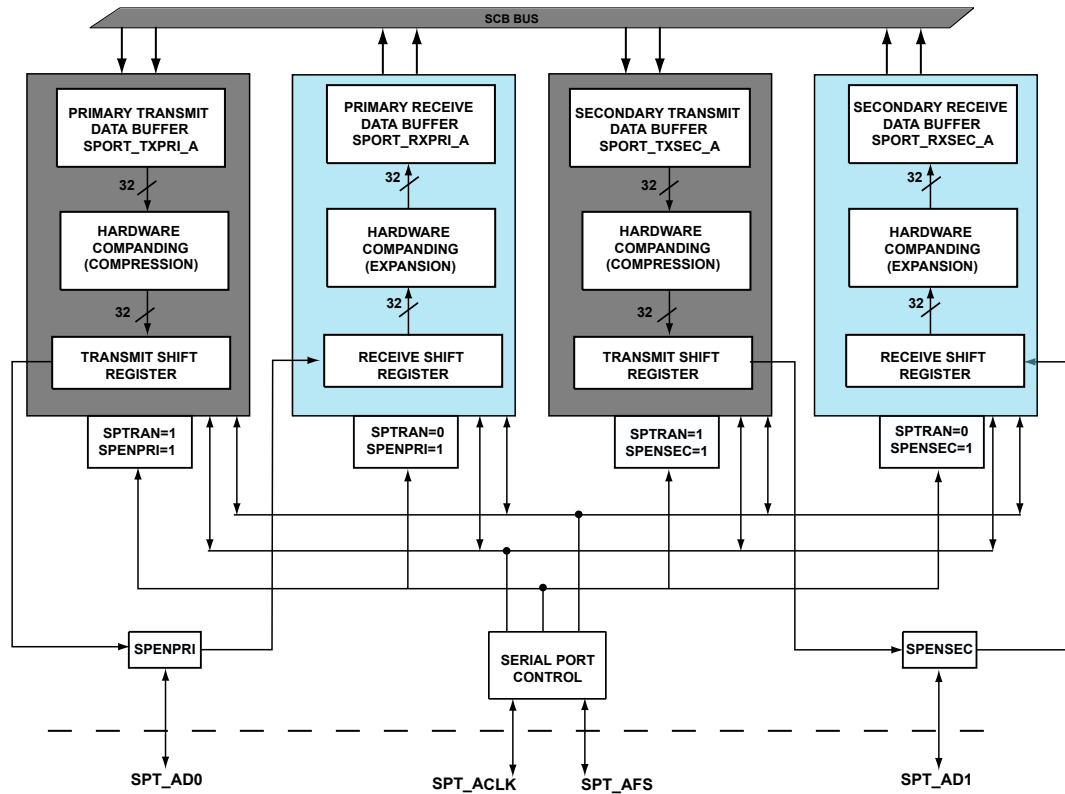


Figure 23-1: Half Serial Port Block Diagram

Architectural Concepts

Each half SPORT (HSPORT) block has its own set of control registers and data buffers, grouped per SPORT module. The HSPORT A and B blocks can be independently configured as either a transmitter or a receiver, with the option to be coupled together internally within the single SPORT top module. Each HSPORT also supports two synchronous bidirectional datapaths, referred to as the primary (D0) and secondary (D1) data lines, as shown in the *Top-Level SPORT Diagram* figure.

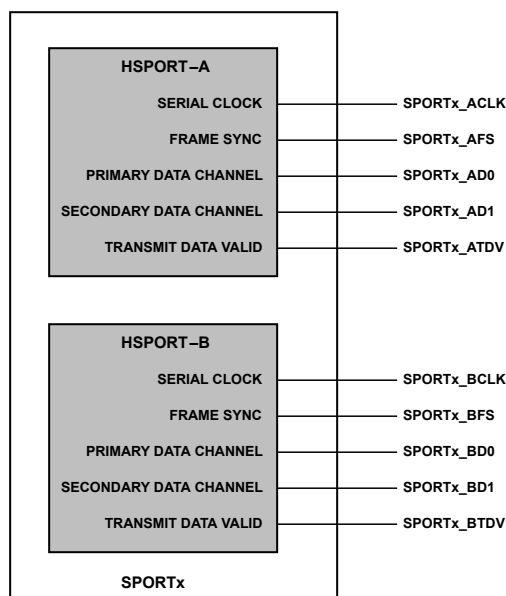


Figure 23-2: Top-Level SPORT Diagram

The `SPORT_CTL_A.SPTRAN` bit controls the direction for both datapaths of the HSPORT. Depending on whether the HSPORT is a transmitter or a receiver, the pair of data signals respectfully transmit or receive data bits synchronously. The dual data signals of each HSPORT cannot transmit and receive the data simultaneously in support of full-duplex operation, however, two HSPORTs can be combined to achieve this.

Serial communications are synchronized to the serial clock signal, where a valid clock pulse must accompany each data bit. Each HSPORT can take its clock from an external source or internally generate it from the processor's system clock using the `SPORT_DIV_A.CLKDIV` clock divisor bit field. Both primary and secondary data channels shift data based on the `SPORT_CLK` rate and the clock polarity defined by the `SPORT_CTL_A.CKRE` bit.

In addition to the serial clock signal, a frame synchronization signal is used to signify the beginning of an individual data word or a multichannel data stream (block of words). Each SPORT can take the frame sync signal from an external source or generate it (`SPORT_FS`), depending on the `SPORT_CTL_A.IFS` bit. An internally generated frame sync is derived from the SPORT clock using the `SPORT_DIV_A.FSDIV` divisor field. Both primary and secondary datapaths start shifting data either synchronous to or one serial clock in advance of detecting/generating a valid frame sync signal, as determined by the `SPORT_CTL_A.LAFS` bit. Various communication protocols for serial data can be emulated according to the frame sync format, and all frame sync options are available whether the signal is generated internally or externally.

NOTE: These SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol.

Multiplexer Logic

The SPORT multiplexing block (SPMUX) is situated between the SPORT hardware block and the processor's pin multiplexing logic. It allows the flexibility to route and share the clock and frame sync signals between the HSPORT A and B halves within each SPORT top module, which can double the data throughput (if both SPORT halves are transmitters or both are receivers) or provide full-duplex capabilities (if one HSPORT is a receiver and the other a

transmitter) without the need to allocate pins for the peripheral or make physical connections outside the processor. The `SPORT_CTL2_A` register is used to configure this loopback feature.

NOTE: Throughout this section, HSPORT A is used as a reference, but all concepts also apply to HSPORT B.

The multiplexing depends on the configuration of the `SPORT_CTL_A.IFS` and `SPORT_CTL_A.ICLK` bits, and the `SPORT_CTL2_A.CKMUXSEL` and `SPORT_CTL2_A.FSMUXSEL` bit settings control the multiplexing. The *Frame Sync Combinations* and *Clock Combinations* tables show the valid combinations for the bit settings.

NOTE: All other settings are illegal. However, hardware does not check or prevent the illegal settings. Ensure that programs use only legal combinations.

The column headers in the *Frame Sync Combinations* table are defined as follows:

- FS Combination = Frame sync combination, referenced in the notes that follow the *Clock Combinations* table.
- HSA_IFS = the setting of the `SPORT_CTL_A.IFS` configuration bit.
- HSB_IFS = the setting of the `SPORT_CTL_B.IFS` configuration bit.
- FSAMUX = the setting of the `SPORT_CTL2_A.FSMUXSEL` configuration bit.
- FSBMUX = the setting of the `SPORT_CTL2_B.FSMUXSEL` configuration bit.

The Routing column in the *Frame Sync Combinations* table defines how the signals are used between the SPORT halves and which pin is used for the frame sync (whether it is an input or an output). Within the column, the inequality characters (\leq and \geq) are used to show the direction of the signal, and the following abbreviations are used (where x = A or B):

- HSx_FI = Frame sync input signal, provided by an external device or the complementing HSPORT.
- HSx_FO = Frame sync output signal.
- SPx_FS = HSPORT's frame sync pin, where the signal is either:
 - provided by an external source and distributed to both HSPORT frame sync signals, or
 - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT frame sync signal.

Table 23-7: Frame Sync Combinations

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
1	0	0	0	0	Native FS Operation
2	0	1	0	0	Native FS Operation
3	1	0	0	0	Native FS Operation
4	1	1	0	0	Native FS Operation

Table 23-7: Frame Sync Combinations (Continued)

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
5	0	0	1	0	HSA_FI \leq SPB_FS; HSB_FI \leq SPB_FS
6	0	1	1	0	HSA_FI \leq HSB_FO \geq SPB_FS
7	0	0	0	1	HSB_FI \leq SPA_FS; HSA_FI \leq SPA_FS
8	1	0	0	1	HSB_FI \leq HSA_FO \geq SPA_FS

The column headers in the *Clock Combinations* table are defined as follows:

- CLK Combination = Clock combination, referenced in the notes that follow the table.
- HSA_ICLK = the setting of the SPORT_CTL_A.ICLK configuration bit.
- HSB_ICLK = the setting of the SPORT_CTL_B.ICLK configuration bit.
- CKAMUX = the setting of the SPORT_CTL2_A.CKMUXSEL configuration bit.
- CKBMUX = the setting of the SPORT_CTL2_B.CKMUXSEL configuration bit.

The Routing column in the *Clock Combinations* table defines how the signals are used between the SPORT halves and which pin is used for the serial clock (whether it is an input or an output). Within the column, the inequality characters (\leq and \geq) are used to show the direction of the signal, and the following abbreviations are used (x = A or B):

- HSx_CI = Serial clock input signal, provided by an external device or the complementing HSPORT.
- HSx_CO = Serial clock output signal.
- SPx_CLK = HSPORT's serial clock pin, where the signal is either:
 - provided by an external source and distributed to both HSPORT serial clock signals, or
 - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT serial clock signal.

Table 23-8: Clock Combinations

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
9	0	0	0	0	Native CLK Operation
10	0	1	0	0	Native CLK Operation
11	1	0	0	0	Native CLK Operation
12	1	1	0	0	Native CLK Operation

Table 23-8: Clock Combinations (Continued)

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
13	0	0	1	0	HSA_CI ≤ SPB_CLK; HSB_CI ≤ SPB_CLK
14	0	1	1	0	HSA_CI ≤ HSB_CO ≥ SPB_CLK
15	0	0	0	1	HSB_CI ≤ SPA_CLK; HSA_CI ≤ SPA_CLK
16	1	0	0	1	HSB_CI ≤ HSA_CO ≥ SPA_CLK

The following is a comprehensive list of the legal combinations for the above described frame sync and clock multiplexing configurations:

- FS Combinations 1–4 are compatible with all CLK Combinations (9–16)
- CLK Combinations 9–12 are compatible with all FS Combinations (1–8)
- FS Combination 5 is only compatible with CLK Combination 13 (and vice versa)
- FS Combination 6 is only compatible with CLK Combination 14 (and vice versa)
- FS Combination 7 is only compatible with CLK Combination 15 (and vice versa)
- FS Combination 8 is only compatible with CLK Combination 16 (and vice versa)

NOTE: Program only the `SPORT_CTL2` register of the HSPORT that is accepting the signal from the other HSPORT. However, be sure to set the `SPORT_CTL_A.CKRE` and `SPORT_CTL_A.LFS` polarity bits to have identical settings between the HSPORTs when making internal connections via the SPMUX block.

Data Types and Companding

The SPORT uses the data type select field `SPORT_CTL_A.DTYPE` bit to specify one of the four data formats supported by serial ports. These formats apply to any of the operating modes of serial port.

Table 23-9: Data Type Bit Field Settings

DTYPE field	SPORT Receiver	SPORT Transmitter
00	Right-justify, zero-fill unused most significant bits	Normal operation
01	Right-justify, sign-extend unused most significant bits	Reserved
10	Expand using μ -law	Compress using μ -law
11	Expand using A-law	Compress using A-law

These formats apply to data words loaded into the SPORT transmit or receive data buffers. The first two data formats (00 and 01 values of `SPORT_CTL_A.DTYPE`) are applicable only when SPORT is configured as receiver.

When SPORT is configured as transmitter, only the significant bits are transmitted (per the field defined in control register). Therefore, the transmit data buffers are not actually zero-filled or sign-extended.

The other two data formats enable the companding logic on the transmit or receive path. Companding (compressing or expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits sent. The SPORTs of the processor support the two most widely used companding algorithms, A-law, and μ -law. The algorithms are performed according to the CCITT G.711 specification.

If selected, companding applies to both the enabled data channels. When enabled as SPORT transmitter, writes to transmit buffer make the content compressed to 8 bits according to algorithm selected. (The content is zero filled to the width of the transmit word.) Similarly, if configured in receive mode, the 8 bits in the receive data buffers expand in right-justified, zero fill format per the algorithm selected. If companding is enabled in multichannel mode, it applies to all the active channels.

The compression for transmit data requires a minimum word length of 8 for proper function. If `SPORT_CTL_A.SLEN` is less than 7, then expansion does not work correctly. Also, if the data value is greater than 13-bit A-law or 14-bit μ -law maximum, it automatically compresses to the maximum value.

NOTE: The processor companding logic supports in-place companding feature. So, companding can be used for debug without enabling SPORT.

Companding as a Function

Since the values in the transmit and receive buffers are companded in place, the SPORT can use the companding hardware without transmitting or receiving data, which can be useful during testing or debugging. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting, use the following procedure:

1. Set the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN = 1`) with both primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI = SPORT_CTL_A.SPENSEC = 0`).
2. Enable companding in the `SPORT_CTL_A.DTYPE` field.
3. Write a 32-bit data word to the transmit buffer.
4. Wait two system clock cycles to allow the SPORT companding hardware to reload the transmit buffer with the companded value. Any instructions that do not access the transmit buffer can be used to cause this delay.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer.

Transmit Path

When the `SPORT_CTL_A.SPTRAN` control bit is set, the HSPORT is in transmit mode. Primary and secondary transmit data paths are then enabled using the `SPORT_CTL_A.SPENPRI` and `SPORT_CTL_A.SPENSEC` bits, respectively. The primary and secondary datapaths are unique and identical, each including its own transmit data buffer, optional companding logic, and transmit shift register.

The data buffer on the primary transmit data path is `SPORT_TXPRI_A`, and the data buffer on the secondary transmit data path is `SPORT_TXSEC_A`. The transmit data buffer and output shift register form a FIFO type of structure. When packing is disabled (`SPORT_CTL_A.PACK = 0`), the SPORT can hold as many as three data words. If packing is enabled (`SPORT_CTL_A.PACK = 1`), the serial port can hold two packed data words at any time.

The transmit data for primary and secondary channels is written to the `SPORT_TXPRI_A` and `SPORT_TXSEC_A` transmit data buffers, respectively. The transmit data buffers can be accessed in core mode through the peripheral bus or in DMA mode through the DMA bus. When a SPORT is configured in transmit mode, the receive paths are deactivated and do not respond to serial clock or frame sync signals. Because the receive data buffers and receive shift registers are also deactivated, reading from an empty and inactive receive data buffer can cause the core to hang indefinitely.

NOTE: Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

This data can optionally be compressed in hardware according to the selected algorithm (μ -law or A-law) and then automatically transferred to the transmit shift register. The shift register, clocked by the `SPORT_ACLK` signal, then serially outputs this data on the `SPORT_AD0` and/or `SPORT_AD1` pins (if both are enabled, these output data bits are transmitted synchronously). If the SPORT uses a framing signal, the `SPORT_AFS` signal indicates the start of the serial word transmission.

When using DMA mode, a single DMA feeds the data buffers of the enabled channels (primary and/or secondary). When using both channels, interleave the data of these channels starting with the primary channel in the transmit buffer.

When the SPORT is configured in non-multichannel mode as a transmitter, the enabled SPORT data pins (`SPORT_AD0` and/or `SPORT_AD1`) are always driven. When a SPORT channel is enabled, data from the transmit data buffer is loaded into the transmit shift register. The shift register then immediately latches the first bit of data (either the LSB or MSB, depending on the `SPORT_CTL_A.LSBF` configuration bit) and drives it to the respective data pin such that it is ready when the frame sync signal asserts. Similarly, if the frame sync period exceeds the serial word length, then the data pins are driven with the first bit of the next word for transmission immediately after the active word completes, and the outputs are held during the inactive serial clock cycles (clock cycles between frame sync pulses).

When the SPORT is configured in multichannel mode, the data pins are driven only during active transmit channels and are always three-stated during inactive channel slots.

The SPORT provides status of transmit data buffers and also error detection logic for transmit errors such as an underrun condition. See the [Error Detection \(Status\) Interrupt](#) section for more details.

Receive Path

When the `SPORT_CTL_A.SPTRAN` bit is cleared, the SPORT is in receive mode. Primary and/or secondary receive data paths can be enabled by setting the `SPORT_CTL_A.SPENPRI` and `SPORT_CTL_A.SPENSEC` configuration bits, respectively. These data paths are unique but identical, each with a receive shift register, optional companding logic, and a receive data buffer.

The data buffer on the primary receive path is `SPORT_RXPRI_A`, and the data buffer on the secondary receive path is `SPORT_RXSEC_A`. The receive data paths act like a three-deep (32-bit words) FIFO because they have two data registers plus an input shift register.

Upon enabling the SPORT data channels, the input shift register shifts in data bits on the `SPORT_AD0` and/or `SPORT_AD1` pins, synchronous to the SPORT clock signal. If the SPORT uses a framing signal, the `SPORT_AFS` signal indicates the beginning of the serial word (or frame) to be received. When an entire word is shifted into the primary and secondary channels, the data can be optionally expanded in hardware according to a selected algorithm (μ -law or A-law) and then automatically transferred to the `SPORT_RXPRI_A` and/or `SPORT_RXSEC_A` data buffers.

The receive data buffers can be read in core mode through the peripheral bus or in DMA mode through the DMA bus. When the SPORT uses DMA mode, a single DMA reads the data buffers of the enabled channels (primary and/or secondary) and interleaves them in memory beginning with the primary channel when both channels are enabled. When using both channels, software must de-interleave the data of these channels.

The SPORT provides the status of receive data buffers and also error detection logic for receive errors such as overflow. See the Error Detection (Status) Interrupt section for more details.

When a SPORT is configured in receive mode, the transmit paths are deactivated and do not respond to serial clock or frame sync signals. As the transmit data buffers and transmit shift registers in the data paths are also deactivated, programs must not try to access them.

NOTE: Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

Operating Modes and Options

The SPORT has a number of operating modes:

- Standard DSP Serial mode
- I²S mode
- Left-Justified mode
- Right-Justified mode
- Multichannel (TDM) mode
- Packed I²S mode

The SPORT halves within a SPORT top module can be independently configured in any of these operating modes unless they are coupled together using SPMUX logic, in which case they must be configured identically. Each half SPORT has its own set of control and data registers and is programmed similarly.

The *Control Bits for SPORT Operating Modes* table lists all the programmable configuration bits in the `SPORT_CTL_A` control register, which combine to determine the overall function and operating mode of the SPORT. The columns are arranged according to the setting of the `SPORT_CTL_A.OPMODE` bit that selects between standard DSP/multichannel modes and the various I²S modes, and the cell contents are defined as follows:

- Yes – bit is programmable for this mode of operation and may be written
- Reserved – bit is not programmable for this mode of operation and must not be written
- = value – bit must be set to this value to enable this mode of operation
- FUNCTION – indicates alternate function for this bit in this mode

NOTE: When changing operating modes, first clear the `SPORT_CTL_A` register before again writing the register with the new configuration settings.

Table 23-10: Control Bits for SPORT Operating Modes

Name (Bit #)	Standard DSP Serial Mode	I ² S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I ² S Mode
SPENPRI (0)	Valid				
DTYPE (2:1)	Valid	Reserved		Valid	
LSBF (3)	Valid	Reserved		Valid	
SLEN (8:4)	Valid				
PACK (9)	Valid				
ICLK (10)	Valid				
OPMODE (11)	= 0	= 1	= 1	= 0	= 1
CKRE (12)	Valid				
FSR (13)	Valid	Reserved			
IFS (14)	Valid				
DIFS (15)	Valid			Reserved	
LFS (16)	Valid	L_FIRST/PLFS		Valid	L_FIRST/PLFS
LAFS (17)	Valid	OPMODE2	Valid	Reserved	
RJUST (18)	Reserved		= 1	Reserved	
FSED (19)	Valid	Reserved		Valid	Reserved
TFIEN (20)	Valid				
GCLKEN (21)	Valid		Reserved		

Table 23-10: Control Bits for SPORT Operating Modes (Continued)

Name (Bit #)	Standard DSP Serial Mode	I ² S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I ² S Mode
SPENSEC (24)			Valid		
SPTRAN (25)			Valid		

Serial Word Length

The SPORT uses the `SPORT_CTL_A.SLEN` field to determine the word length of the serial data to transmit or receive. Each half SPORT can independently handle word lengths up to 32 bits, and the value that must be programmed to the `SPORT_CTL_A.SLEN` field is obtained from:

$$\text{SLEN} = \text{Desired SPORT word length} - 1$$

The minimal word length depends on the selected operating mode. Words smaller than 32 bits are right-justified in the transmit or receive buffers; however, data can be shifted in or out in MSB or LSB first format, as configured by the `SPORT_CTL_A.LSBF` bit. The received word can also be sign-extended or zero-filled when storing the data to processor memory, as governed by the `SPORT_CTL_A.DTYPE` bit.

The *Data Lengths for SPORT Operating Modes* table shows the range of valid word lengths for each of the supported SPORT operating modes.

Table 23-11: Data Lengths for SPORT Operating Modes

Mode	SPORT Word Length (SLEN+1)
Standard DSP Serial	4–32
I ² S	5–32
Left-Justified	5–32
Right-Justified	5–32
Multichannel (TDM)	5–32
Packed I ² S	5–32

NOTE: If the companding feature is enabled on the datapath, it limits the word length settings. See [Data Types and Companding](#) for more details about word lengths required for companding. If more than 32 bits per frame sync are required to transmit or receive, use the multichannel mode to spread the data across numerous continuous channels.

Clock Sample and Drive Edges

The SPORT uses two control signals to sample or drive the serial data:

1. Serial clock (`SPORT_ACLK`) - bit clock for the serial data.
2. Frame sync (`SPORT_AFS`) - divides the incoming data stream into frames.

These control signals can be internally generated or externally provided, as determined by the `SPORT_CTL_A.ICLK` and `SPORT_CTL_A.IFS` bit settings, respectively.

Data and frame syncs can be sampled on the rising or falling edges of the SPORT clock signal, as determined by the `SPORT_CTL_A.CKRE` bit. By default, the `SPORT_CTL_A.CKRE = 0` setting configures the falling edge of the `SPORT_ACLK` signal as the sampling edge for receive data and externally supplied frame syncs. The receive data and frame syncs can be sampled on the rising edges of `SPORT_ACLK` when `SPORT_CTL_A.CKRE = 1`.

NOTE: The SPORT drives transmit data and internal frame sync signals on the opposite serial clock edge of the sampling edge. Be sure to select the same value for `SPORT_CTL_A.CKRE` for transmit and receive functions for any two HSPORTs that are connected together, and always verify the correct polarity for any external device connected to the SPORT.

The *Frame Sync and Data Driven on Rising Edge* figure provides an example of the drive and sample edges when two HSPORTs are connected together, each with `SPORT_CTL_A.CKRE = 0`. In this example, the HSPORT that is configured as the transmitter drives the serial clock and frame sync signals, and both HSPORTs are configured for early, active high frame syncs and a word length of eight bits.

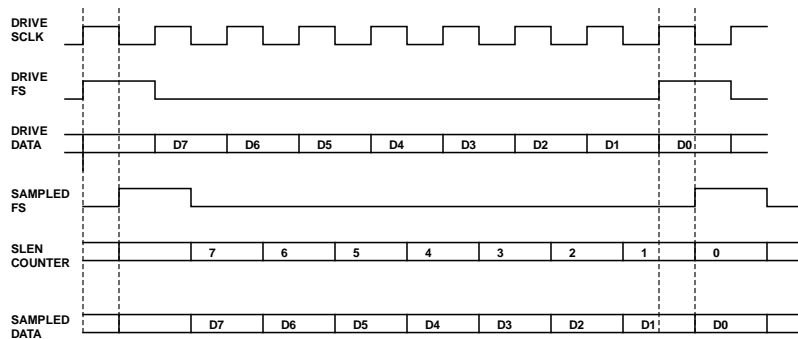


Figure 23-3: Frame Sync and Data Driven on Rising Edge

NOTE: The SCLK in the *Frame Sync and Data Driven on Rising Edge* figure is SCLK.

As shown, the transmitting HSPORT provides the clock and generates the frame sync. Because the HSPORTs are configured for early frame mode, the first bit of data is driven one serial clock later, with subsequent bits being driven on the following rising clock edges in the signal train. When the receiving HSPORT samples the frame sync signal (as indicated in the SAMPLED FS waveform), the `SPORT_CTL_A.SLEN` bit counter is loaded with the `SPORT_CTL_A.SLEN` setting, after which each `SPORT_ACLK` decrements the `SPORT_CTL_A.SLEN` counter until the full word is received. In this figure, the DRIVE FS and SAMPLED FS waveforms show the frame sync required for the next word in a continuous data stream. Note that it is legal for this frame sync to be sampled synchronous to the last bit of the previous data being sampled, as the early frame mode means that the data lags the frame sync by one serial clock cycle. If the frame sync were sampled as asserted before the D0 bit is sampled, the frame sync error is logged in the receiver's status register.

Since the transmitter drives the internally-generated frame sync and data on the rising edge of the serial clock, the receiver must use the falling edge to sample the externally-supplied frame sync and data.

Frame Sync Options

The following sections provide details regarding the programmable aspects of the SPORT frame sync signal. See the specific operating mode sections for additional information regarding frame sync requirements and behavior for each specific operating modes.

Data-Dependent versus Data-Independent Frame Syncs

By default, the generation of a frame sync signal is data-dependent:

- When the SPORT is configured as a transmitter (`SPORT_CTL_A.SPTRAN = 1`), an internally generated transmit frame sync is output when a new data word has been loaded into the channel transmit buffer of the SPORT (by either the core or the DMA engine).
- When the SPORT is configured as a receiver (`SPORT_CTL_A.SPTRAN = 0`), an internally-generated receive frame sync is output only when the receive data buffer is not full.

The data-independent frame sync option, enabled by setting the `SPORT_CTL_A.DIFS` bit, allows for the generation of a periodic framing signal, regardless of the status of the data buffers. When this bit is set, the frame sync output will be continuous and periodic, according to the setting of the `SPORT_DIV_A.FSDIV` field.

Support for Edge-Detected and Level-Sensitive Frame Syncs

The level-sensitive nature of frame sync signals operates well in a noise-free environment. However, if noise corrupts the signals coming into the SPORT, the internal logic can lose synchronization. For example, excessive noise on the frame sync signal may cause the frame sync to be sampled as inactive on the clock edge that it is intended to be synchronous to, but then be sampled at the correct active level one cycle later. Similarly, a noisy clock signal can cause an unintended clock edge, resulting in potential premature sampling of the frame sync signal being applied to the pin.

The *Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync* figure describes a scenario where an external frame sync signal is corrupted due to noise, causing the receiving SPORT module to incorrectly sample the signal. If the frame sync is driven on the rising edge of the serial clock at t_A , the SPORT would normally sample the signal on the falling edge of the serial clock at t_B . Due to the noise, however, the SPORT misses the first edge of the frame sync and instead samples it at t_C .

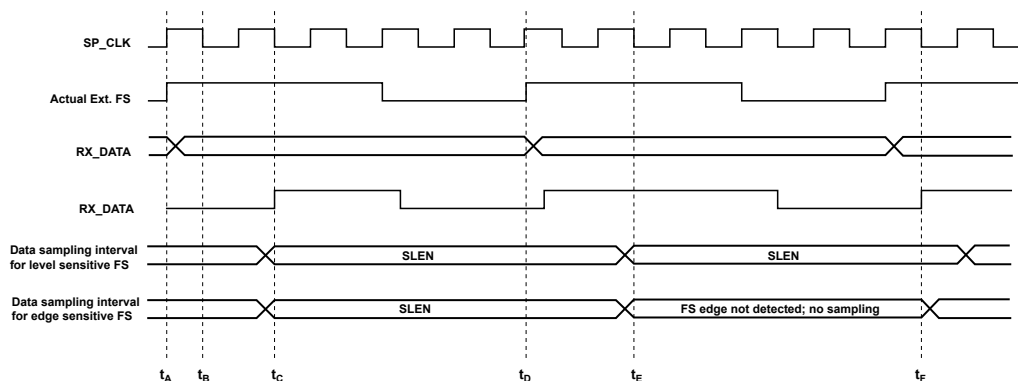


Figure 23-4: Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync

NOTE: SCLK in the *Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync* figure is SCLK.

When the above occurs, the internal word length counter runs for a period equal to the `SPORT_CTL_A.SLEN` field of the control register, but it erroneously expires at t_E rather than at the appropriate point at t_D , thus receiving incorrect data. Further, if a new level-sensitive frame sync edge arrives at time t_D , the SPORT samples this framing signal again at t_E . As such, the frame sync sampling continues to be misaligned with the external data.

To help address this, the SPORT module provides an option to configure the frame sync signal to instead be edge-sensitive via the `SPORT_CTL_A.FSED` configuration bit. When this bit is set with active high frame syncs enabled (`SPORT_CTL_A.LFS = 0`), the rising edge of the frame sync is valid. Conversely, when the frame sync is active low (`SPORT_CTL_A.LFS = 1`), the falling edge is defined to be valid.

NOTE: `SPORT_CTL_A.FSED` is valid only in external frame sync mode. In internal frame sync mode, the setting of this bit is irrelevant and ignored.

In the above example, an edge-sensitive frame sync signal is not detected at t_E because the edge of the framing signal already occurred in the previous cycle (t_D) and there is no new edge to detect at t_E . As a result, the internal word length counter remains idle for this frame, thus ignoring the incorrect data, and the counter correctly resumes operation at t_F when a new frame sync edge is detected.

This activity sets the `SPORT_ERR_A.FSERRSTAT` bit and optionally generates a premature frame sync error interrupt.

Frame sync edge detection is used by default for stereo modes. MCM mode and DSP serial mode choose between edge detection and normal mode of FS detection.

NOTE: When the SPORT is first enabled, an already active externally applied frame sync will not commence operation. The SPORT will wait for a valid change in the frame sync's state from inactive to active before operation begins.

Early versus Late Frame Syncs

Frame sync signals can occur in the same serial clock cycle as the first bit of the data word (late) or one serial clock cycle before the first bit (early), as controlled by the `SPORT_CTL_A.LAFS` bit.

By default, the frame sync signal is configured to be early (`SPORT_CTL_A.LAFS = 0`). The first bit of the transmit data word will be driven one serial clock cycle after the frame sync is asserted (whether sensed externally or internally provided), and the first bit of the receive data word is expected to lag the frame sync by one serial clock cycle. The frame sync is not checked again until the entire word has been transferred.

If data transmission is continuous in early framing mode, then an internally-generated frame sync signal will be asserted (pulsed active for one serial clock cycle) synchronous to the last data bit of the current transfer, as the first bit of the next transfer will be immediately driven in the next serial clock cycle (no clocks are wasted). This event is not a premature frame sync error, so the `SPORT_ERR_A.FSERRSTAT` bit is not set.

The frame sync can alternatively be configured as late (`SPORT_CTL_A.LAFS = 1`), in which case the first bit of the transmit data word is available in the same serial clock cycle that the frame sync is asserted (whether sensed

externally or internally provided), and the first bit of the receive data word is also latched in the same cycle. Serial clock edges latch the receive data bits, but the frame sync signal is checked only during the first bit of each word. Internally generated frame syncs remain asserted for the entire length of the data word in late framing mode.

The *Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)* figure illustrates these concepts.

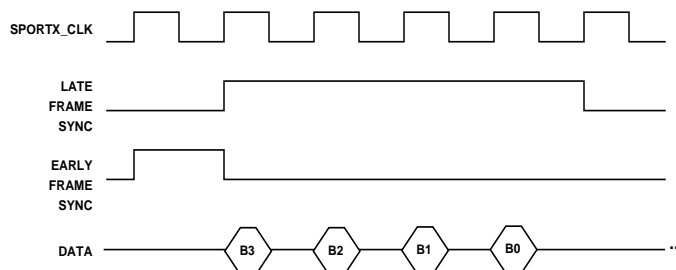


Figure 23-5: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

Framed versus Unframed Frame Syncs

The use of a frame sync signal is optional for SPORT operation, as controlled by the `SPORT_CTL_A.FSR` bit. When the frame sync is configured to be required (`SPORT_CTL_A.FSR = 1`), the data is defined to be framed (a frame sync signal must accompany every data word). To allow continuous transmission from the processor, ensure that a new data word is loaded into the transmit buffer before the ongoing transfer is completed (this is automatically cared for when DMA is used to transmit blocks of data).

Data words can be transferred continuously in what is referred to as unframed data mode, which is appropriate for continuous reception, by setting `SPORT_CTL_A.FSR = 0`. In this configuration, a single frame sync is still required to initiate communication, but it is subsequently unrequired once the communication begins. From that point onward, externally provided frame syncs are ignored and internally generated frame syncs are not driven. The *Framed versus Unframed Data Stream* figure shows the differences in SPORT operation between framed and unframed data modes with the frame sync configured to be early (`SPORT_CTL_A.LAFS = 0`).

NOTE: When DMA is enabled in a mode where frame syncs are not required, chaining can delay DMA requests. DMA requests are not always serviced frequently enough to guarantee continuous unframed data flow. Monitor status bits or check for a SPORT error interrupt to detect underflow or overflow of data.

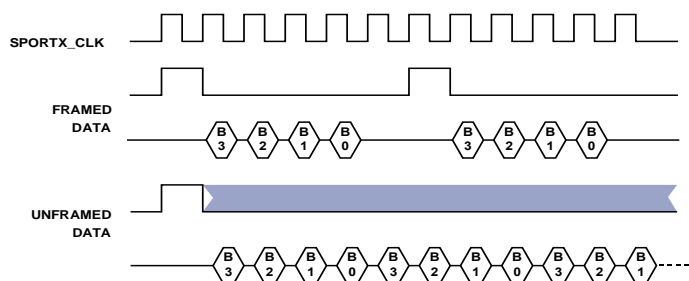


Figure 23-6: Framed versus Unframed Data Stream

Frame Sync Polarity

The framing signals can be active high or active low, as governed by the `SPORT_CTL_A.LFS` bit:

- When `SPORT_CTL_A.LFS = 0`, the corresponding frame sync signal is active high.
- When `SPORT_CTL_A.LFS = 1`, the corresponding frame sync signal is active low.

Active high is the default polarity of the frame sync signal.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external frame sync mode, any frame sync received during an active frame is premature and invalid. When this occurs, the `SPORT_ERR_A.FSERRSTAT` bit is set to indicate the framing error, and an optional error interrupt request can be generated for this event by setting the `SPORT_ERR_A.FSERRMSK` bit.

NOTE: The `SPORT_ERR_A.FSERRSTAT` bit is not set in the presence of uncleared underflow or overflow errors.

Refer to the *Frame Sync Error Detection* figure. The frame sync error bit gets set when an unexpected frame sync occurs during the ongoing data transfer (transmission or reception).

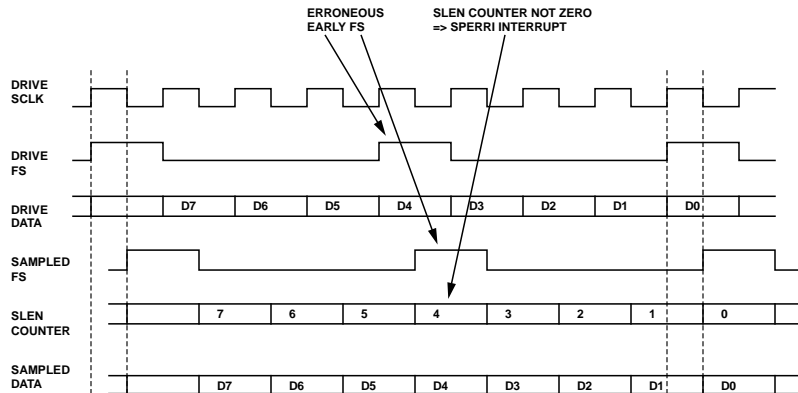


Figure 23-7: Frame Sync Error Detection

NOTE: SCLK in the *Frame Sync Error Detection* figure is SCLK.

Whether a SPORT is receiving or transmitting, its bit count is set to the programmed serial word length when the frame sync is sampled, which is then decremented every subsequent serial clock cycle until the transfer has completed. At this point, the bit count reaches zero and will be reset to the programmed serial length when the next frame sync is sampled. As such, the bit count value is non-zero during an active transfer, and the frame sync error is asserted if a frame sync is sampled when this count is non-zero.

Mode Selection

The SPORT's operating mode is configured in the `SPORT_CTL_A` and `SPORT_MCTL_A` registers. The *SPORT Operating Modes* table provides specific guidance to properly program these SPORT control registers for the desired mode of operation.

Table 23-12: SPORT Operating Modes

Operating Modes	<code>SPORT_CTL_A.OPMODE</code>	<code>SPORT_CTL_A.LAFS</code>	<code>SPORT_CTL_A.RJUST</code>	<code>SPORT_MCTL_A.MCE</code>
Standard DSP Serial	0	Programmable	Reserved	0
I ² S	1	0	Reserved	0
Left-Justified	1	1	Reserved	0
Right-Justified	1	1	1	0
Multichannel	0	Reserved	Reserved	1
Packed I ² S mode	1	Reserved	Reserved	1

The following sections provide detailed information for each of the supported SPORT modes of operation.

Standard DSP Serial Mode

The SPORT can be configured in standard DSP serial mode by clearing the `SPORT_CTL_A.OPMODE` and `SPORT_MCTL_A.MCE` bits. This mode provides great flexibility in terms of programmable options to configure the SPORTs to communicate with various serial devices such as serial data converters and audio codecs. In order to properly connect to such devices, various clocking, framing, and data formatting options are available.

Timing Control Bits

Several bits in the `SPORT_CTL_A` control register define the configuration of the SPORT in standard DSP serial mode:

- **SLEN:** serial word length (4–32 bits)
- **LSBF:** shift LSB or MSB first
- **ICLK:** internally generated or externally provided serial clock
- **CKRE:** sample on rising or falling edge of the serial clock
- **IFS:** internally generated or externally provided frame sync
- **FSR:** framed or continuous operation
- **DIFS:** data-dependent or data-independent frame sync
- **LFS:** active high or active low frame sync
- **LAFS:** frame sync synchronous to data or one clock cycle before it
- **PACK:** 16-bit to 32-bit packing option

- GCLKEN: free-running or gated clock

Clocking Options

In standard DSP serial mode, the SPORTs can either accept an external serial clock or generate one internally, as controlled by the `SPORT_CTL_A.ICLK` bit. For internally generated serial clocks (`SPORT_CTL_A.ICLK = 1`), the `SPORT_DIV_A.CLKDIV` field configures the serial clock rate from the system clock.

The SPORT clock can also be gated, where it is only valid during an active transfer, as controlled by the `SPORT_CTL_A.GCLKEN` bit.

The SPORT clock edge used for driving and sampling of serial data and frame syncs is configured using the `SPORT_CTL_A.CKRE` bit:

- If `SPORT_CTL_A.CKRE = 0`, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If `SPORT_CTL_A.CKRE = 1`, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Stereo Modes

The SPORTs support three widely used stereo modes of operation:

- I²S mode
- Left-Justified mode
- Right-Justified mode

In these modes, the serial data stream consists of left and right channels. The following sections describe these modes in more detail.

Channel Order

The active low frame sync (`SPORT_CTL_A.LFS`) bit is used to determine the polarity of the frame sync signal in the non-stereo modes of operation. For the stereo modes of operation, it instead controls whether the right or left channel is first in the data transfer. The *Channel Order Bit Settings* table shows which word is transmitted or received first, based on the setting of the `SPORT_CTL_A.LFS` bit.

Table 23-13: Channel Order Bit Settings

Mode	<code>SPORT_CTL_A.LFS=0</code>	<code>SPORT_CTL_A.LFS=1</code>
Left-Justified or Right-Justified	Left channel first	Right channel first
I ² S or Packed I ² S	Right channel first	Left channel first

I²S Mode

I²S mode is a commonly used stereo mode, where left and right channel data words are interleaved in the serial data stream and each transition of the frame sync signal is associated with one of the channels. The left channel data is transferred during the low segment of the frame sync signal, and the right channel data is transferred during the high segment of the frame sync signal. As such, the frame sync signal is considered to be a left-right (L/R) clock in this mode.

To set the SPORT up in I²S mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE = 1`
- `SPORT_CTL_A.LFS = 0`
- `SPORT_MCTL_A.MCE = 0`

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to be compliant with the I²S standard, but they can be otherwise configured to support non-standard operation as well:

- **SLEN:** programmable (allowable word lengths are 5–32 bits)
- **LSBF:** set to 0 (MSB first)
- **ICLK:** programmable (serial bit clock can be internally generated or externally provided)
- **IFS:** programmable (serial L/R clock source must match serial bit clock source)
- **LFS:** set to 1 (left channel first)
- **CKRE:** set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS = 1`), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The *Word Select Timing in I²S Mode* figure shows the SPORT timing in I²S mode. The data lags the L/R clock transition by one SCLK cycle, and the transfer begins with the left channel data word first.

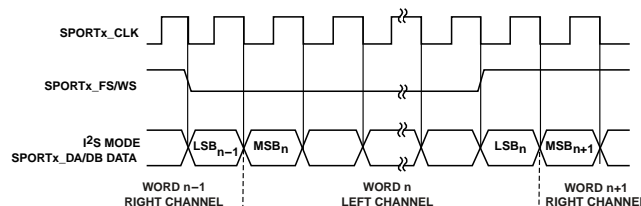


Figure 23-8: Word Select Timing in I²S Mode

Left-Justified Mode

Left-justified mode is a stereo mode subset of the I²S standard. As in I²S mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between left-justified mode and standard I²S mode is that the channel data is driven in the same bit clock cycle as the L/R clock transition (rather than one bit clock cycle later), such that the MSB is synchronous with the leading edge of the frame sync transition.

To set the SPORT up in left-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE = 1`
- `SPORT_CTL_A.LAFS = 1`
- `SPORT_MCTL_A.MCE = 0`

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in left-justified mode, but they can be otherwise configured as well:

- `SLEN`: programmable (allowable word lengths are 5–32 bits)
- `LSBF`: set to 0 (MSB first)
- `ICLK`: programmable (serial bit clock can be internally generated or externally provided)
- `IFS`: programmable (serial L/R clock source must match serial bit clock source)
- `LFS`: set to 0 (left channel first)
- `CKRE`: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS = 1`), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The *Word Select Timing in Left-Justified Mode* figure shows the SPORT timing in left-justified mode. The start of a data sample is synchronous to the L/R clock transition, and the transfer begins with the left channel data word first.

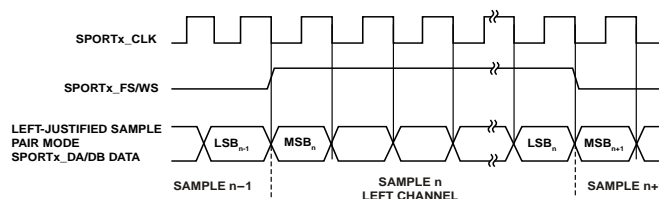


Figure 23-9: Word Select Timing in Left-Justified Mode

Right-Justified Mode

Right-justified mode is a stereo mode subset of the I²S standard. As in I²S mode and left-justified mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between right-justified mode and standard I²S mode is that the LSB of the channel data ends at the point that the L/R clock transitions to frame the next sample (rather than one bit clock cycle after the L/R clock transition).

To set the SPORT up in right-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE = 1`
- `SPORT_CTL_A.RJUST = 1`
- `SPORT_MCTL_A.MCE = 0`

Timing Control Bits

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in right-justified mode, but they can be otherwise configured as well:

- **SLEN:** programmable (allowable word lengths are 5–32 bits)
- **LSBF:** set to 0 (MSB first)
- **ICLK:** programmable (serial bit clock can be internally generated or externally provided)
- **IFS:** programmable (serial L/R clock source must match serial bit clock source)
- **LFS:** set to 0 (left channel first)
- **CKRE:** set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS = 1`), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The *Word Select Timing in Right-Justified Mode* figure shows the SPORT timing in right-justified mode. The transmitter aligns the transmit data such that the last bit of the serial word is sent in the last clock cycle of the L/R clock (frame sync) signal marking the channels.

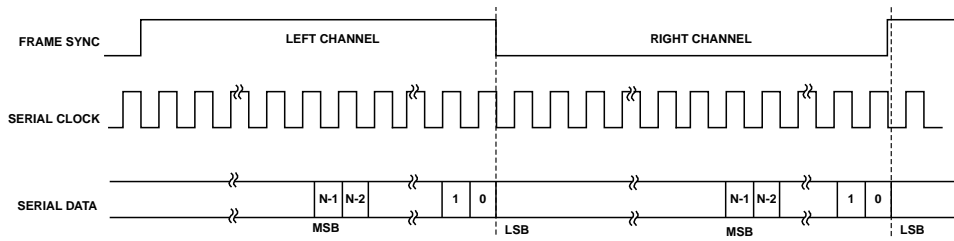


Figure 23-10: Word Select Timing in Right-Justified Mode

NOTE: For some SPORT-compatible ADCs or DACs such as the AD1871, right-justified mode is limited to commonly used ratios such as 64 FS and 128 FS. FS is the sampling frequency of ADCs and DACs, referred to as the SPORT's L/R clock (frame sync) signal.

Consider the SPORT timing for right-justified mode, as shown in the *Timing Comparison Between Different Stereo Modes* figure. The frame sync width is limited to 32 SPORT clock periods (or 32 bits per channel) if:

- the SPORT's frame sync (L/R clock) runs at the FS rate, and
- the SPORT's serial bit clock runs at the 64 FS rate

The limitation applies to the frame sync width of either channel. If the data is confined to 24 bits, the SPORT introduces a $32 - 24 = 8$ -bit clock delay before it starts to transmit or capture data.

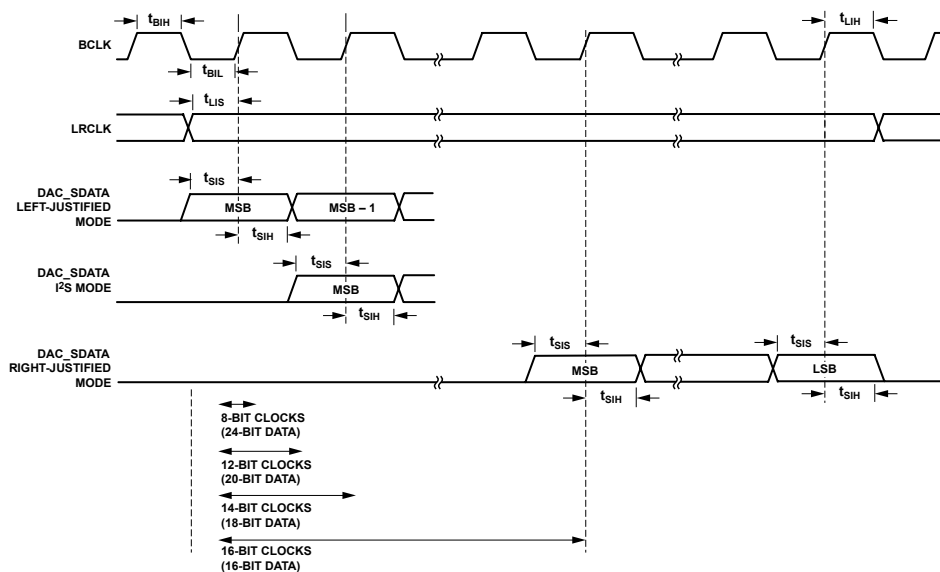


Figure 23-11: Timing Comparison Between Different Stereo Modes

Similarly, to support the 128 FS bit clock frequency, the frame sync width becomes 64 serial bit clock periods per channel. In this case, the delay can be a maximum of 59 bit clocks ($64 - 5$, which is the minimum serial data length in right-justified mode).

The starting point of the first bit is delayed so that the LSB of the serial data aligns properly with the end of the channel. A 6-bit counter is added for this purpose in the stereo mode counter, which is programmed by writing the least significant six bits of the `SPORT_MCTL_A.WOFFSET` field. Though this is a multichannel mode configuration register, the SPORT uses these bits in right-justified mode to configure the offset from the transition of the L/R

clock to where the first data bit must be driven in order to have the end of the last bit align properly with the next L/R clock transition. Software must program this register with the appropriate delay.

Multichannel (TDM) Mode

The multichannel mode of SPORT operation allows the SPORT to communicate as part of a time division multiplexed (TDM) serial system. In TDM communications, a large frame of streamed serial data words is defined to be a particular length. It consists of a specific number of channels, and each channel contains one serial data word of the defined data length. For example, a 24-word block of 24-bit data can be defined to be a window within a frame, having a duration of 576 bit clocks and comprised of 24 continuous channels. The SPORT is configured to transfer on specific channels within a defined window in this frame.

To set the SPORT up in multichannel mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE = 0`
- `SPORT_MCTL_A.MCE = 1`

In multichannel mode, the SPORT can selectively transfer data on up to a maximum window size of 128 continuous channels out of a maximum 1024-channel frame while ignoring all the disabled channels within the window and all the channels outside the window. The SPORT can do any of the following on each channel:

- Transmit data (`SPORT_CTL_A.SPTRAN = 1`)
- Receive data (`SPORT_CTL_A.SPTRAN = 0`)
- Do nothing (during inactive channels)

Channel selection is configured in the half SPORT multichannel select registers (`SPORT_CS0_A - SPORT_CS3_A`) before enabling SPORT operation for multichannel mode. Programming of these registers is especially important in DMA data unpacked mode, since the SPORT data buffers begin operation immediately after the SPORT data lines are enabled. Be sure to enable multichannel operation (set the `SPORT_MCTL_A.MCE` bit) prior to enabling the SPORT itself.

Clocking Options

In multichannel mode, the SPORTs can either accept an external serial clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` bit. For an internally-generated serial clock (`SPORT_CTL_A.ICLK = 1`), the `SPORT_DIV_A.CLKDIV` bit field is used to configure the serial clock rate, as derived from the system clock.

The serial clock edges used to drive and sample data and frame syncs are also configurable using the `SPORT_CTL_A.CKRE` bit:

- If `SPORT_CTL_A.CKRE = 0`, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If `SPORT_CTL_A.CKRE = 1`, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Frame Sync Options

The frame sync signal synchronizes the channels and restarts each multichannel sequence, starting with the channel 0 data word. For internally-generated frame syncs (`SPORT_CTL_A.IFS = 1`), the frame sync period in multichannel mode is defined as:

$$\text{FS period} = [(\text{SPORT_CTL_A.SLEN} + 1) \times \text{number of channels}] - 1$$

The active level for the frame sync signal is also configurable by programming the `SPORT_CTL_A.LFS` bit. Set this bit to make the frame sync an active low signal, and clear it to make it active high.

In multichannel mode, frame sync timing resembles late framing mode (although the `SPORT_CTL_A.LAFS` bit is reserved in this mode). The first bit of the transmit data word is driven and the first bit of the receive data word is sampled in the same serial clock cycle as the frame sync, provided there is no programmed frame delay (`SPORT_MCTL_A.MFD = 0`).

Once the frame sync signal is asserted, word transfers are performed continuously for the duration of the active window, and no further frame syncs are required for different channels within the window. As such, internally-generated frame syncs are always data-independent, and the `SPORT_CTL_A.DIFS` bit is reserved.

Transmit Data Valid (TDV)

Each SPORT features a transmit data valid signal (`SPORT_ATDV`), which is driven high during enabled transmit channels. Because the SPORT output data signals are three-stated during inactive channels, the `SPORT_ATDV` signal signifies when the processor is actively driving the SPORT data outputs, thus serving as an output-enable signal for the data transmit pin(s).

Active Channel Selection Registers

In multichannel mode, the SPORT supports a window size of up to 128 channels for transmitting or receiving data, where it can selectively receive or transmit data in any of these 128 channels. Each channel can be individually enabled or disabled using the multichannel selection registers (`SPORT_CS0_A` to `SPORT_CS3_A`) to select the channels in which to transfer data during a multichannel communication stream. Data words associated with enabled channels are transmitted or received in the respective channels, while disabled channels cause a transmit SPORT to three-state the data output pins and a receive SPORT to ignore the data.

The four 32-bit multichannel selection registers combine to form up to a 128-bit meta-register to accommodate the maximum window size of 128 channels. Setting any bit within these registers enables the associated channel. The 128 channels are sequentially numbered from bit 0 in the `SPORT_CS0_A` register (corresponding to channel 0 of the window) to bit 31 of the `SPORT_CS3_A` register (corresponding to channel 127 of the window). For example, setting bit 13 of the `SPORT_CS1_A` register enables channel number 45 (add 32 for the channels in the `SPORT_CS0_A`). Likewise, setting bit 5 of the `SPORT_CS3_A` register enables channel number 101 (add 96 for the 32 channels in each of the `SPORT_CS0_A`, `SPORT_CS1_A`, and `SPORT_CS2_A` registers).

Multichannel Frame Delay (MFD)

The multichannel frame delay (`SPORT_MCTL_A.MFD`) field specifies the delay in serial bit clocks between the frame sync pulse and the first data bit in the frame. This configurability allows the processor to work with different types of telephony interface devices.

As `SPORT_MCTL_A.MFD` is a 4-bit field, the maximum value allowed for the frame delay is 15 serial clock cycles. When set to 0, the frame sync is concurrent with the first data bit. If `SPORT_MCTL_A.MFD` > 0, a new frame sync can occur during the last channel(s) of a previous frame and still be valid (does not cause a frame sync error).

NOTE: If the required frame delay exceeds 15 serial clocks, use the window offset field (`SPORT_MCTL_A.WOFFSET`) to delay the start of channel 0 in increments of the serial word length, and then adjust `SPORT_MCTL_A.MFD` accordingly. For example, if the serial word length is 12 bits and the desired frame delay is 16 serial clock cycles, set the `SPORT_MCTL_A.WOFFSET` to 1 to insert a 12-bit delay after the frame sync to where the channel 0 data begins, and then program `SPORT_MCTL_A.MFD` to 4 (i.e., 16 - 12).

Window Size (WSIZE)

Select the number of channels used in multichannel operation by programming the 7-bit `SPORT_MCTL_A.WSIZE` field. This field must be set to the actual number of channels minus one (`SPORT_MCTL_A.WSIZE = Number of channels - 1`).

The 10-bit `SPORT_MSTAT_A.CURCHAN` field holds the channel number currently being serviced during multichannel operation.

Window Offset (WOFFSET)

The window offset (`SPORT_MCTL_A.WOFFSET`) field specifies where in the 1024-channel frame to place the start of the active window (up to 128 channels long). A value of 0 specifies no channel offset from the frame sync (channel 0 immediately follows it). Any non-zero value indicates the number of channels that come between the frame sync and the start of channel 0 of the active frame, with 896 (for example, 1024–128) being the largest value that permits using all 128 channels.

As an example, a program could define an active window comprised of eight channels (`SPORT_MCTL_A.WSIZE = 7`) with a window offset of 93 (`SPORT_MCTL_A.WOFFSET = 93`). If configured in this fashion, the 8-channel window that the SPORT will transfer within resides in the channel range from 93 to 100 in the up-to-1024-channel frame.

Do not change the window offset or the number of multichannel slots (`SPORT_MCTL_A.WSIZE`) while the SPORT is enabled. If the combination of the window size and offset place any portion of the window out-of-range relative to the channel counter, none of the channels are enabled.

Companding Selection

Like the other operating modes, companding logic can optionally be applied to serial data (compression logic for transmit mode or expansion logic for receive mode). The two widely used companding algorithms, A-law and μ -law, are selectable using the `SPORT_CTL_A.DTYPE` field.

If companding is enabled, the companding algorithm is applied to both the primary and secondary datapaths. In multichannel mode, companding can be applied to either all or none of the enabled channels (companding cannot be selected on a per-channel basis).

Multichannel DMA Data Packing (MCPDE)

Multichannel DMA data packing and unpacking are enabled using the `SPORT_MCTL_A.MCPDE` bit.

When set, data is packed, and the SPORT expects the data in the DMA buffer to correspond only with enabled SPORT channels. For example, if only channels 1 and 9 are enabled in a 10-channel window (`SPORT_MCTL_A.WSIZE = 9`), the SPORT expects the buffer to be exactly two words in length, where channel 1 is associated with the first element in the buffer and channel 9 is associated with the second.

When cleared, data is unpacked, and the SPORT expects the DMA buffer to have a word for each of the channels in the active window, whether the channel is enabled or not. As such, the DMA buffer size must be exactly the size of the window. Using the same example as the packed case above, if only channels 1 and 9 are enabled in a 10-channel window (`SPORT_MCTL_A.WSIZE = 9`), then the DMA buffer size is ten words. The data at offsets 1 and 9 within the buffer are associated with the data transfers of channels 1 and 9, respectively. The rest of the words in the buffer are unused.

Packed I²S Mode

The SPORT supports a packed I²S mode, which can be used for audio codec communications using multiple channels. This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode).

To set the SPORT up in packed I²S mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE = 1`
- `SPORT_MCTL_A.MCE = 1`

Like multichannel mode, packed I²S mode also supports a maximum of 128 channels, where up to 128 channels of data can be transferred for every transition of the frame sync signal acting as an L/R clock (for example, up to 128 left-channel words transfer during the high portion of the L/R clock, and up to 128 right-channel words transfer during the low portion).

As shown in the *Packed I²S Mode 128 Operation* figure, the packed waveforms are the same as those waveforms used in multichannel mode, except the frame sync is toggled for every frame and emulates I²S mode.

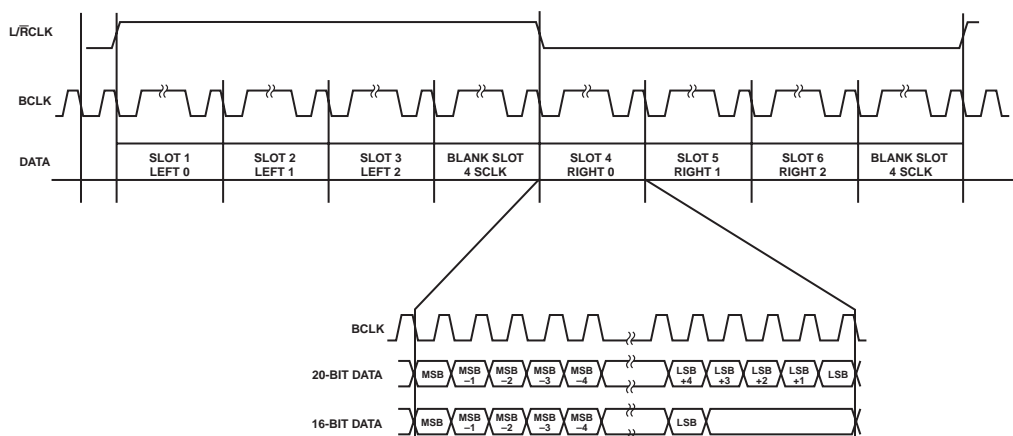


Figure 23-12: Packed I²S Mode 128 Operation

Serial Bit Clock Options

In packed I²S mode, the SPORTs can either accept an external serial bit clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` configuration bit. For an internally-generated serial bit clock (`SPORT_CTL_A.ICLK = 1`), use the `SPORT_DIV_A.CLKDIV` bit field to configure the serial bit clock rate from the system clock.

The serial bit clock edge that is used for sampling or driving data and frame syncs is programmable using the `SPORT_CTL_A.CKRE` bit.

L/R Clock (Frame Sync) Options

The frame sync period in packed I²S mode is defined as:

$$\text{FS period} = [(\text{SPORT_CTL_A.SLEN} + 1) \times \text{number of channels}] - 1.$$

The L/R clock can be supplied externally or internally generated depending on the `SPORT_CTL_A.IFS` bit setting. The logic level of the L/R clock associated with the left and right channel data can be changed using the `SPORT_CTL_A.LFS` configuration bit.

Gated Clock Mode

Some system components such as ADCs and DACs utilize a SPI-compatible protocol for the interface. To communicate with such devices, the SPORT must support a gated clock, where the data valid information is embedded in the clock (for example, the clock only toggles when data is valid). This gated clock feature is enabled using the `SPORT_CTL_A.GCLKEN` bit.

To enable the gated clock mode of operation, program the SPORT to comply with the following requirements.

- Do not enable gated clock functionality in right-justified or multichannel mode
- Gated clock mode has the following requirements for other control bits:

- The serial clock and frame sync signals must have the same source (`SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS`)
- Unframed mode is not supported (`SPORT_CTL_A.FSR` must be set)
- Clear the `SPORT_CTL_A.DIFS` bit in transmit mode; set it in receive mode
- Satisfy the following necessary conditions when gated clock mode is enabled:
 - Seven serial clock cycles are required between enabling the SPORT and the first frame sync. If this requirement is not met, the SPORT can drop the first data (for subsequent data, this requirement is not applicable)
 - For externally-provided clock and frame sync, the frame sync must be inactive during clock synchronization after the SPORT has been enabled
 - For an edge-detected frame sync (`SPORT_CTL_A.FSED = 1`), the frame sync must transition back to the inactive state before the current word transfer is complete (or when the clock is still running). If this requirement is not met, the SPORT does not recognize the next valid frame sync and skips the channel. The SPORT continues to skip the frame syncs until the frame sync transitions back to an inactive state while the clock is active.

Data Transfers and Interrupts

SPORT data can be transferred to or from internal or external memory by two methods:

- Core-driven, single-word transfers
- DMA-driven, multiple-word transfers (optionally with multiple work units)

Core-driven transfers use SPORT interrupts to signal the processor core to perform MMR-based single-word transfers to or from the SPORT data buffers. DMA can be set up to automatically transfer a configurable number of serial words between the SPORT transmit/receive data buffers and memory, and then generate a data completion interrupt request when a work unit or a series of work units completes, thus signaling to the processor core that a block of data has been transferred.

The following sections provide information on core-driven and DMA-driven data transfers.

Data Buffers

When programming the serial port data channels (primary or secondary) as a transmitter by setting `SPORT_CTL_A.SPTRAN = 1`, only the corresponding transmit data buffers (`SPORT_TXPRI_A` and `SPORT_TXSEC_A`) become active. The receive data buffers (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`) remain inactive. Similarly, when the SPORT data channels are programmed for receive operation (`SPORT_CTL_A.SPTRAN = 0`), then only corresponding receive data buffers (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`) are active. Do not attempt to read or write inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results can occur.

Each of these buffers is 32-bit wide (corresponds to maximum serial data word length). When using word lengths less than 32 bits for SPORT operation, the data in these buffers is automatically right-justified. (The LSB bit of data is at the bit 0 location of the buffer). The upper unused bits can be zero-filled or sign-extended depending on `SPORT_CTL_A.DTYPE` field.

Transmit Data Buffers (`SPORT_TXPRI_A` and `SPORT_TXSEC_A`)

When enabled as a transmitter (`SPORT_CTL_A.SPTRAN = 1`), each SPORT half has its own set of transmit data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred to as `SPORT_TXPRI_A` and `SPORT_TXSEC_A` respectively.

These transmit data buffers are 32 bits wide. Load these buffers with the data for transmission on the primary and secondary data channels. The DMA controller loads the data automatically. Or, the program running on the processor core loads the data manually.

Together with the output shift register, transmit data buffers act like a two-location FIFO. If data packing is disabled (`SPORT_CTL_A.PACK = 0`), the transmit path can hold as many as three data words. If data packing is enabled (`SPORT_CTL_A.PACK = 1`), it can hold two packed data words at any given time.

When the transmit shift register becomes empty (transfer out all the bits of previous word), data in the transmit data buffer is automatically loaded into it. An interrupt occurs when the output transmit shift register has been loaded, signifying that the transmit data buffer is empty and ready to accept the next word. This interrupt does not occur when serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not write to the inactive secondary transmit data buffer and conversely. If the core keeps writing to the inactive buffer, the status of that transmit buffer becomes full. This state can cause the core to hang indefinitely, since data is never transmitted to the output shift register.

Receive Data Buffers (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`)

When enabled as receiver (`SPORT_CTL_A.SPTRAN = 0`), each SPORT half has its own set of receive data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred as `SPORT_RXPRI_A` and `SPORT_RXSEC_A` respectively. Together with input shift register, the receive data buffers act like a three-location FIFO, as the receive path has two data registers.

These receive data buffers are the 32 bits wide. These buffers are automatically loaded from the receive shift register when a complete word has been received into it. An interrupt occurs when the receive data buffer is loaded, signifying that new data is available in the receive data buffer and is ready to read. This interrupt does not occur when the serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not read from the inactive secondary receive data buffer and conversely. If the core keeps reading from the inactive buffer, the status of that receive buffer becomes empty. This state can cause the core to hang indefinitely since new data is never received through the input shift register.

Data Buffer Status

The SPORT provides status information about its primary and secondary data buffers through the `SPORT_CTL_A.DXSPRI` and `SPORT_CTL_A.DXSSEC` bits, respectively. It also provides error status information through the corresponding `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` bits, respectively. Depending on the `SPORT_CTL_A.SPTRAN` bit setting, these bits reflect the status of either the pair of transmit (`SPORT_TXPRI_A` and `SPORT_TXSEC_A`) or receive (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`) buffers, indicating whether the buffer is full, partially full, or empty.

When attempting to read from an empty receive buffer or write to a full transmit buffer, the SPORT delays access until the buffer is ready, potentially resulting in excessive MMR bus response times. To avoid this when doing core-driven transfers, always check the buffer status to determine if the access can be made. The SPORT updates the status bits in the `SPORT_CTL_A` register during reads and writes by the core processor.

NOTE: These status bits are updated during reads and writes from the core processor even when the SPORT is disabled.

Two complete 32-bit words can be stored in the receive buffer while a third word shifts in. Therefore, almost three complete words can be received without the receive buffer being read before an overflow occurs. After receiving the third word completely, the shift register contents overwrite the second word, which will occur if the first word has not yet been read by the processor core or the DMA controller. This receive overflow condition is flagged through the error status bits of the `SPORT_CTL_A` register on the last bit of the third word.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less, the received data words can be packed into a 32-bit word. Similarly, if the SPORT is configured as a transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted can be unpacked into 16-bit words. The `SPORT_CTL_A.PACK` bit is used to select this packing or unpacking feature.

When `SPORT_CTL_A.PACK = 1`, two consecutive received words are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This packing method applies to both receive (packing) and transmit (unpacking) operations. In this case, the transmit and receive interrupt requests are generated for the 32-bit packed words, not for each 16-bit word.

NOTE: When 16-bit received data is packed into 32-bit words and stored in normal word space in the processor's internal memory, the 16-bit words can be read or written using short word space addressing.

Single-Word (Core) Transfers

The SPORTs can transmit or receive individual data words with interrupt requests occurring as each data word is transferred. When a SPORT is enabled with the corresponding DMA channel disabled, interrupt requests are generated when:

- a complete word has been received in the receive data buffer or

- the transmit data buffer is not full

When performing core transfers, be sure to access only those buffers that are associated with enabled datapaths, as governed by the transfer direction (`SPORT_CTL_A.SPTRAN`) bit and the primary/secondary (`SPORT_CTL_A.SPENPRI/SPORT_CTL_A.SPENSEC`) data enable bits. If inactive SPORT data buffers are read from or written to by the core while the SPORT is enabled, the core can hang. For example, if a half SPORT is programmed to be a transmitter and the core reads from one of the receive buffers associated with that half SPORT, the core can hang as if it were reading an empty buffer that is active and awaiting new data to arrive. Because this is a transmitting HSPORT, that data will never arrive, thus locking the core up until the SPORT is reset. To avoid such a situation, be sure to check the status of the appropriate data buffer before attempting a core access to it by interrogating the `SPORT_CTL_A.DXSPRI` or `SPORT_CTL_A.DXSSEC` status bits.

DMA Transfers

Direct memory access (DMA) provides a mechanism for transferring an entire block of serial data before an interrupt is generated. The processor's on-chip DMA controller automatically handles the DMA transfer, thus allowing the processor core to run in parallel until the entire block of data is transferred. When the interrupt request occurs, a service routine can then process the entire block of data (rather than react to single words), thus significantly reducing overhead.

Each half SPORT has a dedicated DMA channel that serves both the primary and secondary datapaths. When configured as a transmitter (`SPORT_CTL_A.SPTRAN = 1`) with both the primary and secondary datapaths enabled (`SPORT_CTL_A.SPENPRI = SPORT_CTL_A.SPENSEC = 1`), the DMA channel requires that the source DMA buffer interleave the data beginning with the primary channel, as it will alternately load to the primary and secondary transmit data buffers once it is enabled. The complementary operation is true in receive mode (`SPORT_CTL_A.SPTRAN = 0`) when both datapaths are enabled, as the DMA channel alternately reads from the primary and secondary receive data buffers and interleaves them in the destination DMA buffer. As such, software must de-interleave the data corresponding to the primary and secondary channels from the receive DMA buffer.

If the SPORT is configured in stereo mode, the same DMA channel handles both the left and right channels of both datapaths (primary and/or secondary). Therefore, for a transmit DMA with only one datapath enabled, the source buffer must be populated such that the left- and right-channel data is interleaved. If both datapaths are enabled, the DMA channel alternately loads to the primary and secondary transmit data buffers once it is enabled. As such, the interleaving requirement is for the primary left-channel data to be followed by the secondary left-channel data, then the primary right-channel data, and finally the secondary right-channel data. The complementary operation is true in receive mode, where the DMA channel alternately reads from the primary and secondary receive data buffers and interleave them in the destination DMA buffer. For the stereo modes of operation, the destination DMA buffer is interleaved as left-right data for a single data input. If both datapaths are enabled, the destination DMA buffer is written with the primary and secondary left-channel data followed by the primary and secondary right-channel data. As such, software must de-interleave the primary and secondary left- and right-channel data from the receive DMA buffer, as defined by this scheme.

Since both the primary and secondary datapaths share the single DMA channel, each half SPORT has a single interrupt request for data completion, as well as an error interrupt request. The DMA controller can generate an interrupt request at the end of a chain of DMA work units (when using multiple descriptors) or at the end of individual DMA work unit.

The SPORT DMA channels are assigned a higher priority than all the other DMA channels (for example, the SPI port). Having higher priority causes the SPORT DMA transfers to execute first when multiple DMA requests occur in the same cycle. The SPORT DMA channels are numbered and prioritized in the DMA channel list table in the DMA chapter.

Although the most efficient DMA transfers execute with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by `SPORT_CTL_A.SLEN` field). If the serial data length is 16 bits or smaller, two pieces of data can be packed into 32-bit words for each DMA transfer, as selected by setting the `SPORT_CTL_A.PACK` bit. When this bit is set, the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit word. For more information, see the [Data Buffer Status](#) section.

NOTE: The SPORT DMA channel can access both internal memory and external memory of the processor without any core overhead.

Data Transfer Interrupt

Each half SPORT features a data transfer interrupt request that is shared by both the primary and secondary data channels in both transmit and receive modes. To determine the source of the data transfer interrupt request, applications can check the primary and secondary data buffer status bits (`SPORT_CTL_A.DXSPRI` and `SPORT_CTL_A.DXSSEC`, respectively).

When using core-driven transfers, this interrupt's meaning depends on the direction of the SPORT:

- As transmitter (`SPORT_CTL_A.SPTRAN = 1`) - the transmit data buffer is empty
- As receiver (`SPORT_CTL_A.SPTRAN = 0`) - new data is available in the receive data buffer

NOTE: When data packing is enabled (`SPORT_CTL_A.PACK = 1`), the core-driven transmit and receive interrupt requests are generated for 32-bit packed words, not for each 16-bit word.

In both cases, the interrupt request can be used to signal the core that an individual transfer has completed. For transmit operations, it indicates that the transmit data buffer can be safely loaded (either the buffer is already empty or the last data has moved from the data buffer to the shift register). For receive operations, it indicates that new data has arrived and can be read (or must be read before a subsequent word overwrites it).

When the SPORT is configured to use DMA to move data between memory and the peripheral (the most generic way to use dedicated DMA for sport data transfers), the same data transfer interrupt request instead indicates the completion of the transfer of a block of serial data (rather than a single word). When DMA is used, the DMA count register must be initialized to specify the number of words to transfer. This count decrements after each DMA transfer on the channel, and the data transfer interrupt request signal is asserted when the word count reaches zero (for example, a DMA work unit has finished).

For transmit DMA, the interrupt request is raised when the last word in the DMA work unit is loaded from the source memory to the HSPORT FIFO. This interrupt request can signal to the core that a new DMA work unit can be configured or that other software threads can now run. The transmit interrupt request can optionally be deferred until the last word of the work unit has fully shifted out of the shift register (see the Transfer Finish Interrupt (TFI) section for details).

For receive DMA, the interrupt request is raised when the last word is loaded to the destination memory. In addition to that described for transmit DMA, this interrupt request also serves as an indication to the core that there is a buffer of newly acquired data that is ready to be processed.

See the DMA chapter for further details regarding enabling of the DMA interrupt requests associated with the various modes of DMA operation.

NOTE: As a single DMA channel services both the primary and secondary datapaths associated with the SPORT, there is a single DMA completion interrupt request.

Transfer Finish Interrupt (TFI)

When configured for transmit DMA (`SPORT_CTL_A.SPTRAN = 1`), the data transfer interrupt request gets generated by the DMA engine itself when it decrements its count register upon loading the last element from memory to the HSPORT hardware. Alternately, the SPORT can use a Transmit Finish Interrupt (TFI) to signal the actual end of the transmission (for example, when the last bit of the last data word of the buffer has shifted out of the SPORT to the system) by setting the `SPORT_CTL_A.TFIEN` bit. When this bit is set, then DMA signal that would normally assert the data transfer interrupt request instead signals the SPORT that the DMA work unit is complete. The SPORT then waits until all the data in the FIFO is shifted out (including the transmit shift register) and asserts the TFI interrupt request upon completion.

NOTE: To enable this functionality in the DMA engine, be sure to configure the interrupt type field in the DMA configuration register for Peripheral interrupt. See the DMA chapter for further details.

Error Detection (Status) Interrupt

In addition to the dedicated data transfer interrupt request, each half SPORT also features an optional error status interrupt request that can be triggered when error conditions occur relative to data or frame syncs associated with the half SPORT.

Data-related errors depend on the direction of the SPORT and reflect overflow or underflow conditions, which are depicted in the `SPORT_CTL_A` control register as read-only sticky bits `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` (for the primary and secondary channels, respectively).

- When the SPORT is configured as a transmitter, these bits provide transmit data buffer underflow status. When the frame sync signal occurs when the transmit data buffer is empty, the underflow bit corresponding with the offending transmit data buffer is set, as the SPORT will transmit data whenever it detects a valid frame sync signal, whether new data is present or not.
- When the SPORT is configured as a receiver, these bits provide receive overflow status. When a channel receives new data while the receive buffer is already full, the new data overwrites the existing data, thus causing

an overflow. When this occurs, the overflow bit corresponding with the offending receive data buffer is set, as the SPORT receives data whenever it detects a valid frame sync signal, whether there is room in the receive buffer or not.

Each half SPORT also features an error register ([SPORT_ERR_A](#)), which is the source for the assertion of the described data-related error status bits. When a data-related error occurs on the primary or secondary datapaths, the error is logged in the `SPORT_ERR_A.DERRPSTAT` or `SPORT_ERR_A.DERRSSTAT` bits, respectively. To enable these status bits to generate the HSPORT status interrupt request in the SEC, the corresponding `SPORT_ERR_A.DERRPMSK` and `SPORT_ERR_A.DERRSMSK` bits must be set (for the primary and secondary datapaths, respectively).

The `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` channel error status bits are sticky read-only bits that can be cleared in two ways:

- Reset the error detection logic by disabling the channel associated with the error condition (clear the `SPORT_CTL_A.SPENPRI` or `SPORT_CTL_A.SPENSEC` control bit).
- Clear the source of the interrupt by writing-1-to-clear the `SPORT_ERR_A.FSERRSTAT`, `SPORT_ERR_A.DERRPSTAT`, or `SPORT_ERR_A.DERRSSTAT` status bits.

In addition to data-related errors, [SPORT_ERR_A](#) also tracks frame sync errors in the `SPORT_ERR_A.FSERRSTAT` status bit. Similar to the data-related errors, the frame sync error can be enabled as a source for raising the error status interrupt request via the SEC by setting the `SPORT_ERR_A.FSERRMSK` bit. A frame sync error occurs when the frame sync is detected prematurely, as explained in the [Premature Frame Sync Error Detection](#) section.

A frame sync error is not detected in the following cases:

- When there is no active transmit or receive data, and the frame sync pulse occurs due to noise on the input signal – if there is no active transfer, a noise-induced frame sync pulse is valid.
- If there is an active underflow or overflow error – frame sync errors cannot be detected because the SPORT error logic does not run after one of the data errors has occurred and remains unserved.
- When the frame sync pulse does not meet minimum timing requirements – if the frame sync pulse is shorter than a SPORT clock period, there is no guarantee that it gets sampled at all and may go unnoticed.

Grouping of SPORTs

For some applications, enabling multiple SPORTs need to be synchronized so that all of them start and/or end at the same time. It may also be required to generate/receive a single interrupt/trigger for the DMA transfer of all the SPORTs in the same group.

The [DAI_GBL_SP_EN](#) register is used to control enabling/disabling. The [DAI_GBL_INT_EN](#) register can be used to control interrupt/trigger generation/reception by multiple SPORTs in a group.

NOTE: All references to SPORT 0-3 in the context of DAI1 must be read as SPORT4-7.

SPORT Enable Grouping

The figure shows the block diagram of possible group enabling options for the SPORTs in DAI0 and DA1.

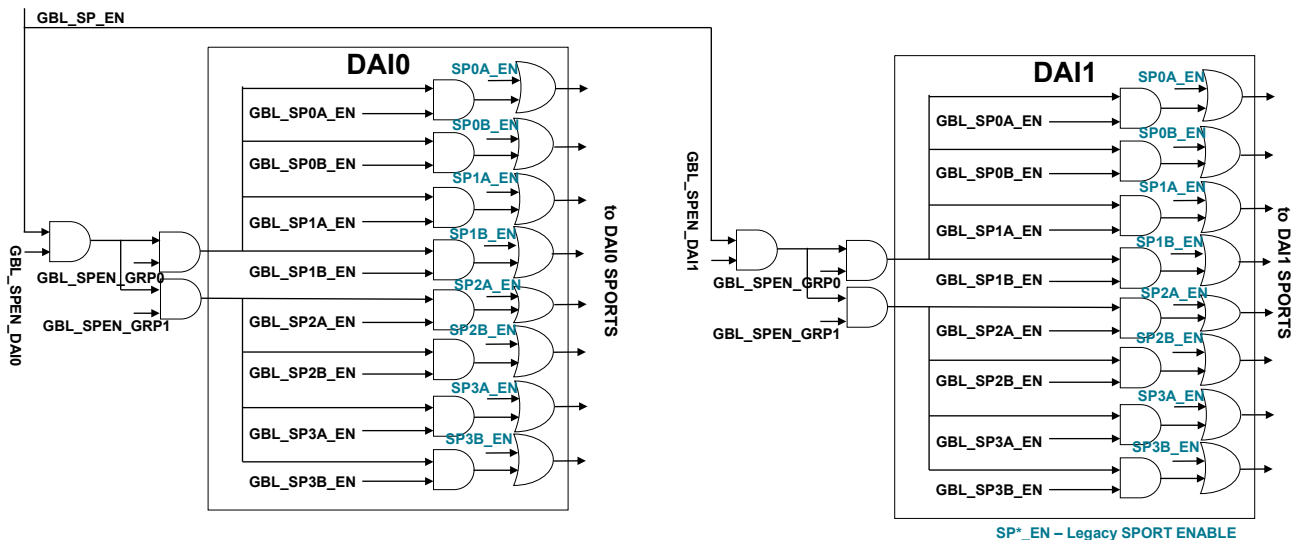


Figure 23-13: Group Enabling Options in DAI0 and DA1

It shows one of the channels in each half sport, same is applicable for other channel as well. For example, GBL_SP0A_EN means GBL_SP0A_PC_EN or GBL_SP0A_SC_EN.

Each DAI has a `DAI_GBL_SP_EN` register. To enable the SPORTs grouped within the same DAI at the same time, write to the corresponding `DAI_GBL_SP_EN` register is required. If a group of SPORTs in different DAIs has to be enabled at the same time, write to the `DAI_GBL_SP_EN.GBL_SP_EN` field is required. This bit is reserved for the DA11.

The following group enable options are possible:

- Up to 16 channels in 8 half SPORTs in DAI0
- Up to 16 channels in 8 half SPORTs in DA11
- Up to 8 channels in 4 half SPORTs in each DAI
- Up to 32 channels in 16 half SPORTs (8 half SPORTs in DAI0 and 8 half SPORTs in DA11)

NOTE: While grouping SPORTs across DAIs, two interrupts (one per DAI) are generated and managed by software. Grouping can be done only once as the `DAI_GBL_SP_EN.GBL_SP_EN` field is used.

NOTE: The grouping of SPORTs and its interrupts/triggers are two separate entities. If the interrupts and triggers must be grouped similar to the SPORT grouping, use the group interrupt/trigger enable register.

SPORT Interrupt/Trigger Grouping

Each DAI has a `DAI_GBL_INT_EN` register. Each of these registers has two group interrupt enable bits `DAI_GBL_INT_EN.GRP0_INT_EN` and `DAI_GBL_INT_EN.GRP1_INT_EN`, two group trigger (master)

enable bits `DAI_GBL_INT_EN.GRP0_TRG_EN` and `DAI_GBL_INT_EN.GRP1_TRG_EN`. There are individual `GRPx_SPyINT_EN` bits for each group to enable both interrupt/trigger for up to 8 channels (4 half SPORTs). Enable these bits as per the grouping in the `DAI_GBL_SP_EN` register.

NOTE: To enable or to disable the SPORT interrupt/trigger grouping feature, the individual sport interrupt/trigger enable bits must be programmed along with the respective group interrupt/trigger enable bits in the `DAI_GBL_INT_EN` register.

Regarding grouping of SPORT interrupts/triggers:

- All the interrupts/triggers in a single group are ANDed and a single interrupt/trigger goes to SEC/TRU.
- To generate group interrupt/trigger, all the SPORTs in the corresponding group should be configured to generate interrupt (`DMA_CFG.INT`)/trigger (`DMA_CFG.TRIG`).
- The interrupts/triggers generated by the SPORTs in a group can either be used as a single group interrupt/trigger (enabled by the `DAI_GBL_INT_EN` register) or the individual SPORT DMA done interrupts/triggers.
- As there are no separate status registers for global interrupts, to clear the global interrupts, clear (W1C) the individual SPORT `DMA_STAT.IRQDONE` bits. Theoretically, the global interrupt gets cleared by clearing the `DMA_STAT.IRQDONE` bit of at least one SPORT in the group. However, it is recommended to clear the `DMA_STAT.IRQDONE` bits of all the SPORTs in the group to achieve a clean state for the next global interrupt generation.
- The `DAI_GBL_INT_EN.GRP1_TRG_EN/DAI_GBL_INT_EN.GRP0_TRG_EN` bits enables group master trigger functionality for SPORTs in a group.

SPORT Programming Model

The following sections provide programming guidance for setting up the SPORTs for use in an application:

- [Initializing Core-Driven \(Non-MCM\) Transfers](#)
- [Initializing Multichannel Transfers](#)
- [Using DMA for SPORT Transfers](#)
- [Using Companding as a Function](#)

Initializing Core-Driven (Non-MCM) Transfers

The following programming model applies to all of [Standard DSP Serial Mode](#), [I²S Mode](#), [Left-Justified Mode](#), and [Right-Justified Mode](#) for core-driven transfers. More steps are required to properly initialize the SEC to service the SPORT interrupts (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Clear the `SPORT_CTL_A` and `SPORT_MCTL_A` configuration registers.

ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multi-channel logic) is fully reset before attempting to reprogram it.

2. Optionally program the `SPORT_DIV_A` clock divisor register.

ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the [Serial Clock](#) and [Frame Sync](#) sections.

3. Program the `SPORT_CTL_A` primary configuration register.

ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the [Operating Modes and Options](#) section for details). Do not set the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` buffer enable bits in this step.

4. Optionally program the `SPORT_CTL2_A` secondary configuration register.

ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the [Multiplexer Logic](#) section for details).

5. Optionally program the `SPORT_ERR_A` error register.

ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt is desired (see the [Error Detection \(Status\) Interrupt](#) section for details).

6. For Right-Justified mode only, program the `SPORT_MCTL_A.WOFFSET` field.

ADDITIONAL INFORMATION: In Right-Justified mode, this field serves as the delay count (DCNT) required to align the LSB of each stereo channel with the L/R clock transition and must be programmed manually (see the [Right-Justified Mode](#) section for details).

7. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate.

8. Write data to be transmitted to the transmit buffer (`SPORT_TXPRI_A` and/or `SPORT_TXSEC_A`) or read data that has been received from the receive buffer (`SPORT_RXPRI_A` and/or `SPORT_RXSEC_A`).

ADDITIONAL INFORMATION: These accesses are typically performed in the context of an interrupt service routine. See the SEC chapter for further information. Do not attempt to read or write inactive data buffers. If the core attempts to access inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Initializing Multichannel Transfers

When in [Multichannel \(TDM\) Mode](#) or [Packed I²S Mode](#), the SPORT is in a multichannel operational mode. Follow the steps below to properly initialize the SPORT for multichannel modes of operation. More steps are required to properly (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate register changes, this example also applies to half SPORT B.

1. Clear the `SPORT_CTL_A` and `SPORT_MCTL_A` registers.

ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multichannel logic) is fully reset before attempting to reprogram it.

2. Optionally program the `SPORT_DIV_A` clock divisor register.

ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the [Serial Clock](#) and `SPORT_CTL2_A` sections.

3. Program the `SPORT_CS0_A` - `SPORT_CS3_A` channel select registers.

4. Program the `SPORT_MCTL_A` multichannel configuration register.

ADDITIONAL INFORMATION: The SPORT supports many multichannel options. For more information, see the [Multichannel \(TDM\) Mode](#) section. Do not set the `SPORT_MCTL_A.MCE` enable bit in this step.

5. Program the `SPORT_CTL_A` primary configuration register.

ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the [Operating Modes and Options](#) section for details). Do not set the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` buffer enable bits in this step.

6. Optionally program the `SPORT_CTL2_A` secondary configuration register.

ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the [Multiplexer Logic](#) section for details).

7. Optionally program the `SPORT_ERR_A` error register.

ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt request is desired (see the [Error Detection \(Status\) Interrupt](#) section for details).

8. Set the `SPORT_MCTL_A.MCE` bit to enable multichannel mode.

9. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate. DMA mode is recommended for multichannel modes of operation. For more information, see the [Using DMA for SPORT Transfers](#) programming model.

Using DMA for SPORT Transfers

DMA is supported in all SPORT operating modes ([Standard DSP Serial Mode](#), [I²S Mode](#), [Left-Justified Mode](#), [Right-Justified Mode](#), [Multichannel \(TDM\) Mode](#) or [Packed I²S Mode](#)). To enable DMA operation with the SPORT, execute the steps described in this section after initializing and enabling the SPORT. Instead of using the single word read or write operations described in the referenced programming models, the DMA engine automates accesses to the enabled SPORT data buffers.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, it also applies to half SPORT B.

1. Follow the guidance in the multichannel ([Initializing Multichannel Transfers](#)) or non-multichannel ([Initializing Core-Driven \(Non-MCM\) Transfers](#)) programming models to properly initialize and enable the SPORT hardware.
2. Prepare the data buffers in memory.

ADDITIONAL INFORMATION: Ensure that the DMA buffer is defined according to the [DMA Transfers](#) section. For the multichannel modes of operation, be sure to also consider the setting of the `SPORT_MCTL_A.MCPDE` bit, as described in the [Multichannel DMA Data Packing \(MCPDE\)](#) section.

3. Initialize and enable the DMA channel allocated for the SPORT, as described in the Direct Memory Access (DMA) chapter.

Using Companding as a Function

The data in the transmit and receive buffers are actually companded in place. As such, the following programming model can be used to exercise the companding hardware without transferring data, which is useful for test/debug purposes.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Configure the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN=1`) with both the primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI=0` and `SPORT_CTL_A.SPENSEC=0`).
2. Enable the desired companding scheme in the `SPORT_CTL_A.DTYPE` field.
3. Write a 32-bit word to one of the transmit buffers.
4. Wait two system clock cycles.

ADDITIONAL INFORMATION: This delay is required to allow the SPORT companding hardware to reload the transmit buffer with the companded result. Any instructions that do not access the transmit buffer can be used to cause this delay.

5. Read the 8-bit compressed value from the transmit buffer written above.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (`SPORT_CTL_A.SLEN`).

Programming Global SPORT Groups

Complete the following steps to program global SPORT groups.

NOTE: To complete all of the SPORT group transfers simultaneously, all of the SPORTs in the group must have a similar data/clock/frame sync configuration.

1. For each SPORT, program the following registers for the desired SPORT mode of operation.
 - `SPORT_DIV_A/SPORT_DIV_B`
 - `SPORT_CTL2_A/SPORT_CTL2_B`
 - `SPORT_CTL_A/SPORT_CTL_B`
 - `SPORT_MCTL_A/SPORT_MCTL_B`
 - `SPORT_CS0_A/SPORT_CS0_B`
 - `SPORT_CS1_A/SPORT_CS1_B`
 - `SPORT_CS2_A/SPORT_CS2_B`
 - `SPORT_CS3_A/SPORT_CS3_B`

ADDITIONAL INFORMATION: The sport enable control bits (in `SPORT_CTL2_A/SPORT_CTL2_B` registers) must be cleared (disabled).

2. For a global group trigger as the master, program each SPORT DMA in the group as the master trigger.
3. Program the sport enable and group enable bits in the `DAI_GBL_SP_EN` register for the desired global sport grouping.

ADDITIONAL INFORMATION: The grouping of SPORTs (enable and disable) and the associated interrupts/triggers are separate activities. If the interrupts and triggers must be grouped similar to the SPORTs (enable and disable) grouping, use the `DAI_GBL_INT_EN` register.

4. Program the `DAI_GBL_INT_EN` register to enable the grouped interrupt or trigger generation for the SPORT group.

ADDITIONAL INFORMATION: To enable the SPORT grouped interrupt and trigger feature, each sport interrupt or trigger enable bit must be programmed with the respective group interrupt and trigger enable bits in the `DAI_GBL_INT_EN` register.

5. Program the `DAI_GBL_SP_EN.GBL_SP_EN` bit to enable the global sport groups at the same instance.

Disabling Global SPORT Groups

Complete the following steps to disable global SPORT groups.

1. Use the individual SPORT DMA status register to clear the group interrupt. There is no specific status register for global interrupts.

2. Disable the global SPORT group interrupt and trigger bits for the group in the `DAI_GBL_INT_EN` register.

ADDITIONAL INFORMATION: To disable the SPORT grouped interrupt and trigger feature, each sport interrupt and trigger enable bit must be programmed with the respective group interrupt and trigger enable bits in the `DAI_GBL_INT_EN` register.

3. Disable the global individual SPORT enable and the global SPORT group enable bits in the `DAI_GBL_SP_EN` register.

4. Disable the DMAs for the SPORTs when the transfer is complete.

ADSP-2156x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 23-14: ADSP-2156x SPORT Register List

Name	Description
<code>SPORT_CS0_A</code>	Half SPORT 'A' Multichannel 0-31 Select Register
<code>SPORT_CS0_B</code>	Half SPORT 'B' Multichannel 0-31 Select Register
<code>SPORT_CS1_A</code>	Half SPORT 'A' Multichannel 32-63 Select Register
<code>SPORT_CS1_B</code>	Half SPORT 'B' Multichannel 32-63 Select Register
<code>SPORT_CS2_A</code>	Half SPORT 'A' Multichannel 64-95 Select Register
<code>SPORT_CS2_B</code>	Half SPORT 'B' Multichannel 64-95 Select Register
<code>SPORT_CS3_A</code>	Half SPORT 'A' Multichannel 96-127 Select Register
<code>SPORT_CS3_B</code>	Half SPORT 'B' Multichannel 96-127 Select Register
<code>SPORT_CTL2_A</code>	Half SPORT 'A' Control 2 Register
<code>SPORT_CTL2_B</code>	Half SPORT 'B' Control 2 Register
<code>SPORT_CTL_A</code>	Half SPORT 'A' Control Register
<code>SPORT_CTL_B</code>	Half SPORT 'B' Control Register

Table 23-14: ADSP-2156x SPORT Register List (Continued)

Name	Description
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multichannel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

Half SPORT 'A' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the `SPORT_CS0_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

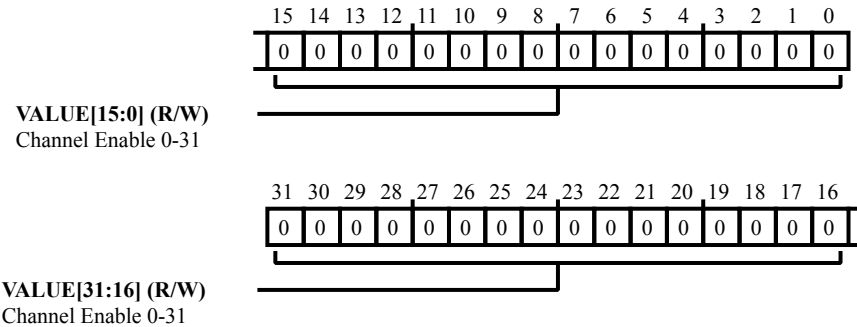


Figure 23-14: `SPORT_CS0_A` Register Diagram

Table 23-15: `SPORT_CS0_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'B' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the `SPORT_CS0_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

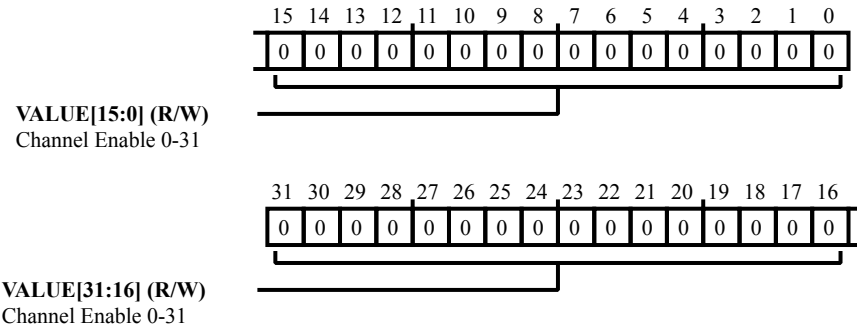


Figure 23-15: `SPORT_CS0_B` Register Diagram

Table 23-16: `SPORT_CS0_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'A' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the `SPORT_CS1_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

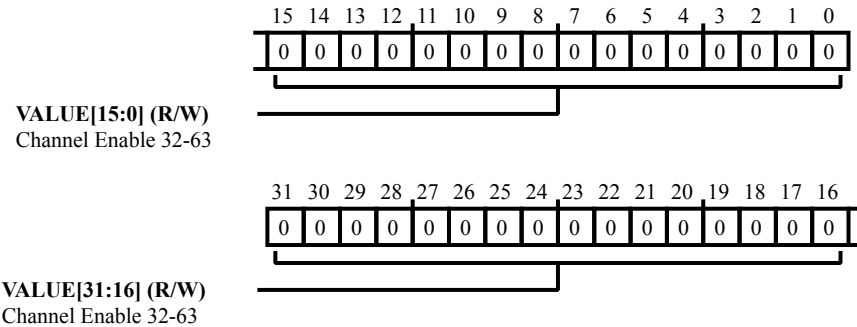


Figure 23-16: `SPORT_CS1_A` Register Diagram

Table 23-17: `SPORT_CS1_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'B' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the `SPORT_CS1_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

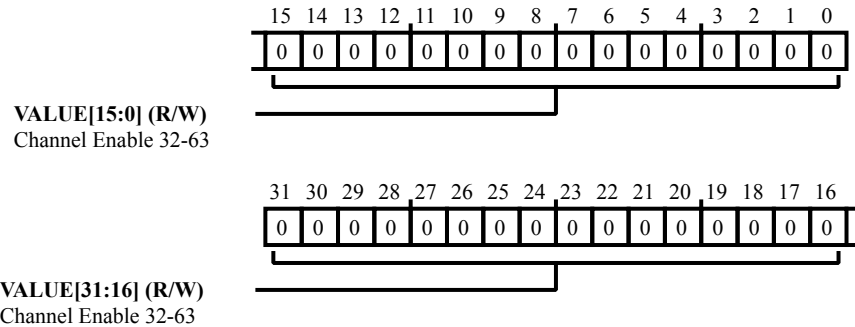


Figure 23-17: `SPORT_CS1_B` Register Diagram

Table 23-18: `SPORT_CS1_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'A' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the `SPORT_CS2_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

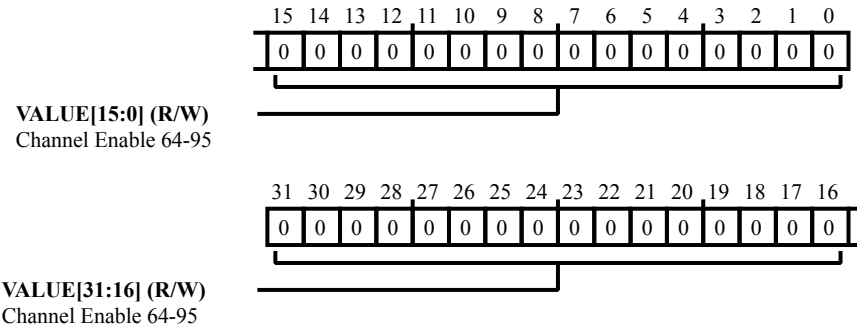


Figure 23-18: `SPORT_CS2_A` Register Diagram

Table 23-19: `SPORT_CS2_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'B' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the `SPORT_CS2_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

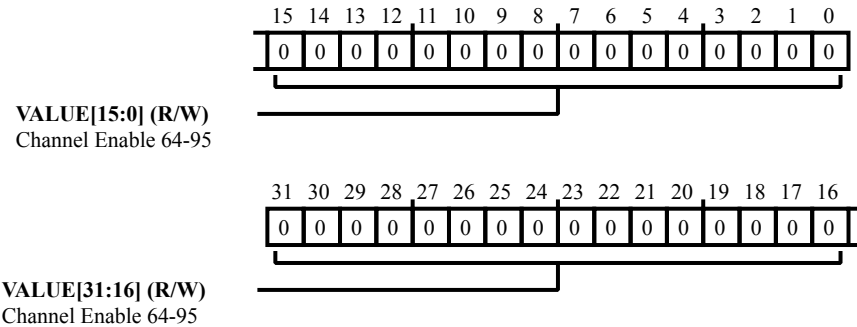


Figure 23-19: `SPORT_CS2_B` Register Diagram

Table 23-20: `SPORT_CS2_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'A' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the `SPORT_CS3_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

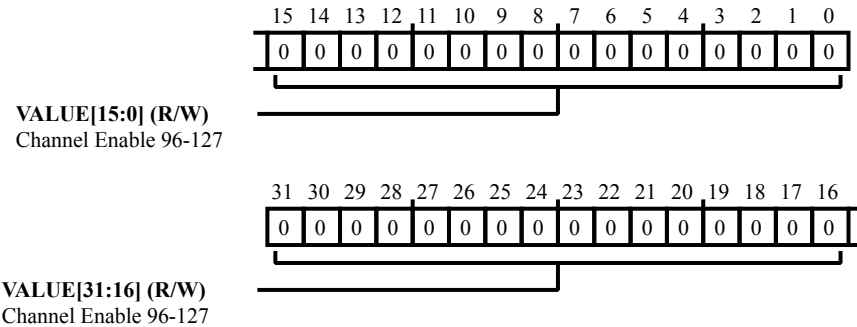


Figure 23-20: `SPORT_CS3_A` Register Diagram

Table 23-21: `SPORT_CS3_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'B' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the `SPORT_CS3_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

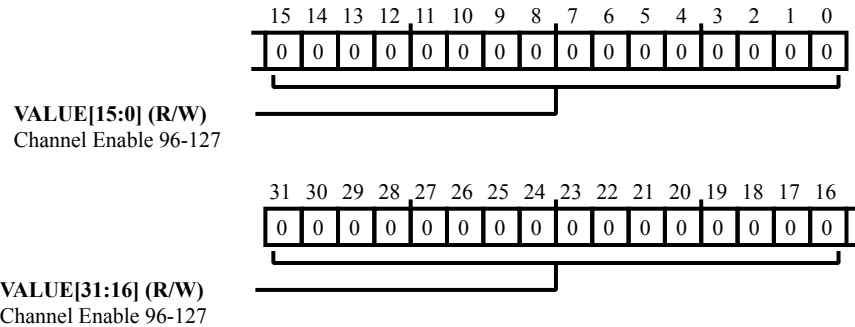


Figure 23-21: `SPORT_CS3_B` Register Diagram

Table 23-22: `SPORT_CS3_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'A' Control 2 Register

The `SPORT_CTL2_A` register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

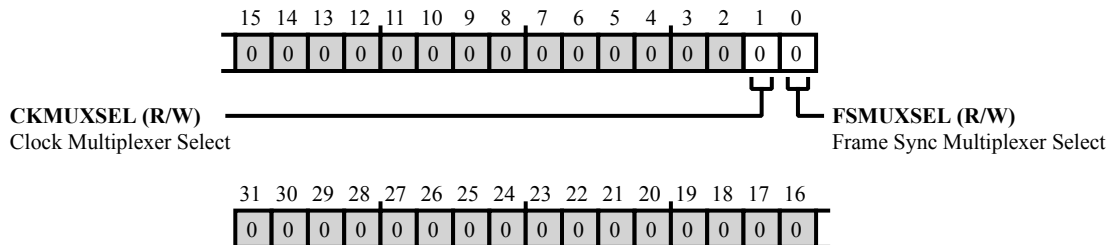


Figure 23-22: `SPORT_CTL2_A` Register Diagram

Table 23-23: `SPORT_CTL2_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The <code>SPORT_CTL2_A.CKMUXSEL</code> bit enables multiplexing of the half SPORT's serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the <code>SPORT_CTL2_A.CKMUXSEL</code> bit is enabled, half SPORT 'A' uses <code>SPORT_BCLK</code> instead of <code>SPORT_ACLK</code> .
		0 Disable serial clock multiplexing
		1 Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The <code>SPORT_CTL2_A.FSMUXSEL</code> bit enables multiplexing of the half SPORT's frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the <code>SPORT_CTL2_A.FSMUXSEL</code> bit is enabled, half SPORT 'A' uses <code>SPORT_BFS</code> instead of <code>SPORT_AFS</code> .
		0 Disable frame sync multiplexing
		1 Enable frame sync multiplexing

Half SPORT 'B' Control 2 Register

The `SPORT_CTL2_B` register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

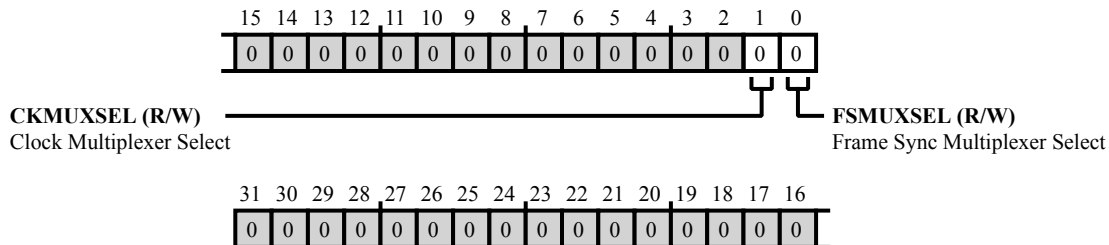


Figure 23-23: SPORT_CTL2_B Register Diagram

Table 23-24: SPORT_CTL2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The <code>SPORT_CTL2_B.CKMUXSEL</code> bit enables multiplexing of the half SPORT's serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the <code>SPORT_CTL2_B.CKMUXSEL</code> bit is enabled, half SPORT 'B' uses <code>SPORT_ACLK</code> instead of <code>SPORT_BCLK</code> .
		0 Disable serial clock multiplexing
		1 Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The <code>SPORT_CTL2_B.FSMUXSEL</code> bit enables multiplexing of the half SPORT's frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the <code>SPORT_CTL2_B.FSMUXSEL</code> bit is enabled, half SPORT 'B' uses <code>SPORT_AFS</code> instead of <code>SPORT_BFS</code> .
		0 Disable frame sync multiplexing
		1 Enable frame sync multiplexing

Half SPORT 'A' Control Register

The `SPORT_CTL_A` register contains transmit and receive control bits for SPORT half 'A', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the `SPORT_CTL_A` register vary depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

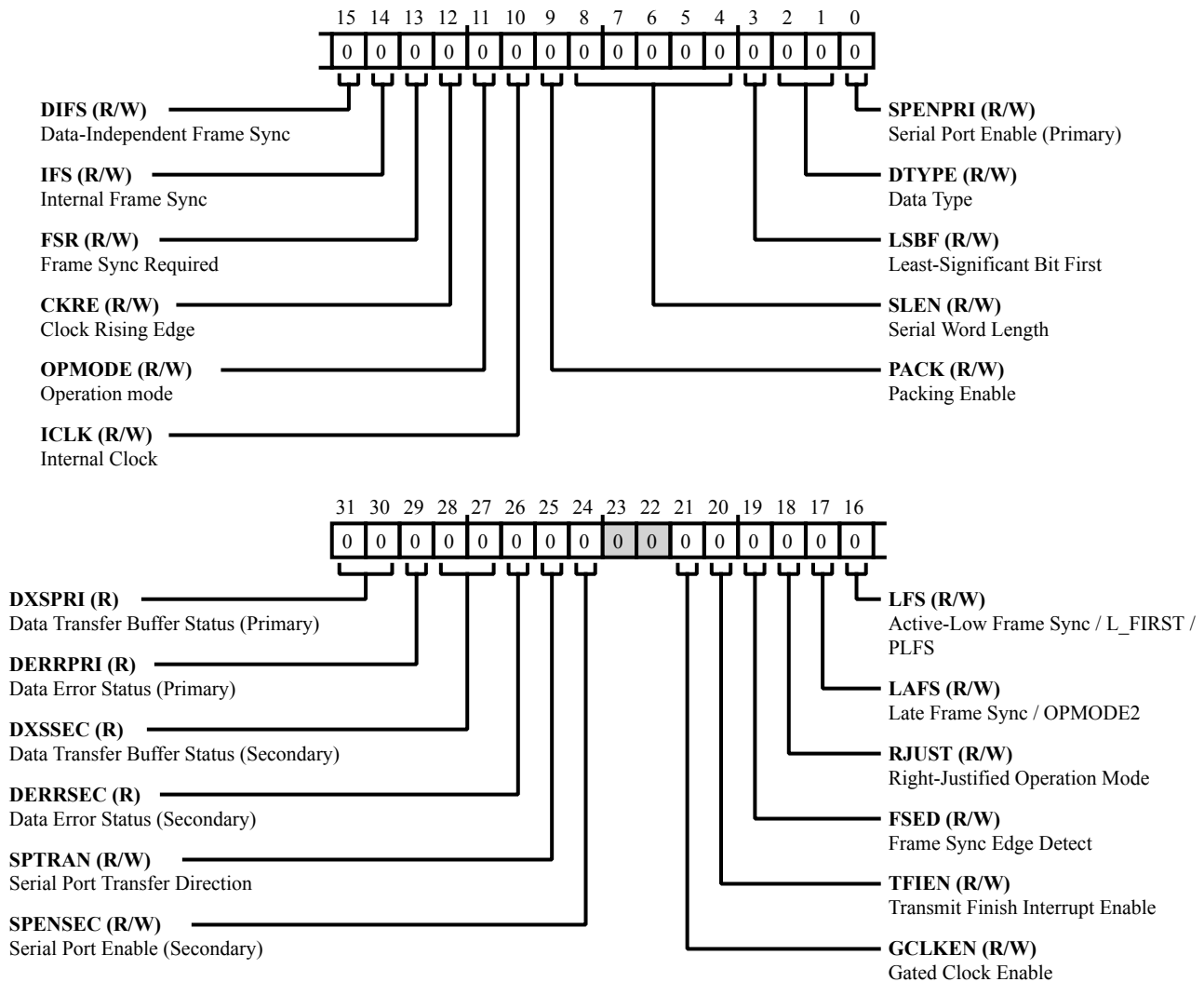


Figure 23-24: `SPORT_CTL_A` Register Diagram

Table 23-25: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The <code>SPORT_CTL_A.DXSPRI</code> bit field indicates the status of the half SPORT's primary channel data buffer.
		0 Empty
		1 Reserved
		2 Partially full
		3 Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The <code>SPORT_CTL_A.DERRPRI</code> bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the <code>SPORT_CTL_A.FSR</code> bit =1, the <code>SPORT_CTL_A.DERRPRI</code> bit indicates whether the <code>SPORT_AFS</code> signal (from an internal or external source) occurred while the <code>SPORT_TXPRI_A</code> data buffer was empty (during transmit) or the <code>SPORT_RXPRI_A</code> data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the <code>SPORT_AFS</code> signal. It is important to note that, as a receiver, the <code>SPORT_CTL_A.DERRPRI</code> bit indicates when the channel has received new data while the <code>SPORT_RXPRI_A</code> receive buffer is full. This new data overwrites existing data. If the <code>SPORT_CTL_A.FSR</code> bit =0, the <code>SPORT_CTL_A.DERRPRI</code> bit is set whenever the SPORT is required to transmit while the <code>SPORT_TXPRI_A</code> transmit buffer is empty. It is also set whenever the SPORT is required to receive while the <code>SPORT_RXPRI_A</code> receive buffer is full. The SPORT clears the <code>SPORT_CTL_A.DERRPRI</code> bit if the <code>SPORT_ERR_A.DERRPSTAT</code> bit is cleared.
		0 No error
		1 Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The <code>SPORT_CTL_A.DXSSEC</code> bit field indicates the status of the half SPORT's secondary channel data buffer.
		0 Empty
		1 Reserved
		2 Partially full
		3 Full

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The <code>SPORT_CTL_A.DERRSEC</code> bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the <code>SPORT_CTL_A.FSR</code> bit =1, the <code>SPORT_CTL_A.DERRSEC</code> bit indicates whether the <code>SPORT_AFS</code> signal (from an internal or external source) occurred while the <code>SPORT_TXSEC_A</code> data buffer was empty (during transmit) or the <code>SPORT_RXSEC_A</code> data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the <code>SPORT_AFS</code> signal. It is important to note that, as a receiver, the <code>SPORT_CTL_A.DERRSEC</code> bit indicates when the channel has received new data while the <code>SPORT_RXSEC_A</code> receive buffer is full. This new data overwrites existing data. If the <code>SPORT_CTL_A.FSR</code> bit =0, the <code>SPORT_CTL_A.DERRSEC</code> bit is set whenever the SPORT is required to transmit while the <code>SPORT_TXSEC_A</code> transmit buffer is empty. It is also set whenever the SPORT is required to receive while the <code>SPORT_RXSEC_A</code> receive buffer is full. The SPORT clears the <code>SPORT_CTL_A.DERRSEC</code> bit if the <code>SPORT_ERR_A.DERRSSTAT</code> bit is cleared.
		0 No error
		1 Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The <code>SPORT_CTL_A.SPTRAN</code> bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the <code>SPORT_ACLK</code> and <code>SPORT_AFS</code> pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the <code>SPORT_ACLK</code> and <code>SPORT_AFS</code> pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.
		0 Receive
		1 Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The <code>SPORT_CTL_A.SPENSEC</code> bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.
		0 Disable
		1 Enable

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	GCLKEN	Gated Clock Enable. The <code>SPORT_CTL_A.GCLKEN</code> bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (<code>SPORT_CTL_A.OPMODE = 0</code> or <code>1</code>). This bit is ignored when the half SPORT is in right-justified mode (<code>SPORT_CTL_A.RJUST = 1</code>) or multichannel mode (<code>SPORT_MCTL_A.MCE = 1</code>). When the <code>SPORT_CTL_A.GCLKEN</code> bit is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).
		0 Disable
		1 Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The <code>SPORT_CTL_A.TFIEN</code> bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the <code>DMA_CFG.INT</code> configuration. When enabled (<code>SPORT_CTL_A.TFIEN = 1</code>), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (<code>SPORT_CTL_A.TFIEN = 0</code>), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).
		0 Last word sent (DMA count done) interrupt
		1 Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The <code>SPORT_CTL_A.FSED</code> bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The <code>SPORT_CTL_A.FSED</code> bit may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (<code>SPORT_CTL_A.FSED = 0</code>), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.
		0 Level detect frame sync
		1 Edge detect frame sync

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	RJUST	Right-Justified Operation Mode. The <code>SPORT_CTL_A.RJUST</code> bit enables the half SPORT (if <code>SPORT_CTL_A.OPMODE = 1</code>) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the <code>SPORT_MCTL_A.WOFFSET</code> field). For information about appropriate delay selections, see the SPORT operating modes section.
		0 Disable
		1 Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (<code>SPORT_CTL_A.OPMODE = 0</code>) or in right-justified mode (<code>SPORT_CTL_A.RJUST = 1</code>), the <code>SPORT_CTL_A.LAFS</code> bit selects whether the half SPORT generates a late frame sync (<code>SPORT_AFS</code> during first data bit) or generates an early frame sync signal (<code>SPORT_AFS</code> during serial clock cycle before first data bit). When the half SPORT is in I ² S / left-justified mode (<code>SPORT_CTL_A.OPMODE = 1</code>), the <code>SPORT_CTL_A.LAFS</code> bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I ² S mode. When the half SPORT is in multichannel mode (<code>SPORT_MCTL_A.MCE = 1</code>), the <code>SPORT_CTL_A.LAFS</code> bit is reserved.
		0 Early frame sync (or I ² S mode)
		1 Late frame sync (or left-justified mode)
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (<code>SPORT_CTL_A.OPMODE = 0</code>), the <code>SPORT_CTL_A.LFS</code> bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I ² S / packed / left-justified mode (<code>SPORT_CTL_A.OPMODE = 1</code>), the <code>SPORT_CTL_A.LFS</code> bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.
		0 Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I ² S/packed mode) or left channel first (left-justified mode)
		1 Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I ² S/packed mode) or right channel first (left-justified mode)

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	DIFS	Data-Independent Frame Sync. The <code>SPORT_CTL_A.DIFS</code> bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by the <code>SPORT_DIV_A.FSDIV</code> bit. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the <code>SPORT_CTL_A.DIFS</code> bit is automatically set when the half SPORT is in packed or multichannel modes.
		0 Data-dependent frame sync
		1 Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The <code>SPORT_CTL_A.IFS</code> bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.
		0 External frame sync
		1 Internal frame sync
13 (R/W)	FSR	Frame Sync Required. The <code>SPORT_CTL_A.FSR</code> bit selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I ² S / packed / left-justified mode (<code>SPORT_CTL_A.OPMODE = 1</code>) or is in multichannel mode (<code>SPORT_MCTL_A.MCE = 1</code>).
		0 No frame sync required
		1 Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The <code>SPORT_CTL_A.CKRE</code> bit selects the rising or falling edge of the <code>SPORT_ACLK</code> clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the <code>SPORT_ACLK</code> . Also, note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for the <code>SPORT_CTL_A.CKRE</code> bit. This programming drives the internally-generated signals on one edge of <code>SPORT_ACLK</code> and samples the received signals on the opposite edge.
		0 Clock falling edge
		1 Clock rising edge

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	OPMODE	Operation mode. The SPORT_CTL_A.OPMODE bit selects whether the half SPORT operates in DSP standard/multichannel mode or operates in I ² S/packed/left-justified mode. The mode selection affects the operation of the SPORT_CTL_A.LAFS and SPORT_CTL_A.LFS bits. Also, the SPORT_CTL_A.OPMODE bit enables or disables operation of the SPORT_CTL_A.GCLKEN, SPORT_CTL_A.FSED, SPORT_CTL_A.RJUST, SPORT_CTL_A.DIFS, SPORT_CTL_A.FSR, and SPORT_CTL_A.CKRE bits.
		0 DSP standard/multichannel mode
		1 I ² S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_A.OPMODE =0), the SPORT_CTL_A.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_ACLK clock signal, and SPORT_ACLK is an output. The SPORT_DIV_A.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_ACLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.
		0 External clock
		1 Internal clock
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_A.PACK bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.
		0 Disable
		1 Enable

Table 23-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8:4 (R/W)	SLEN	<p>Serial Word Length.</p> <p>The <code>SPORT_CTL_A.SLEN</code> bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is:</p> $\text{SPORT_CTL_A.SLEN} = (\text{serial word length in bits}) - 1$ <p>For DSP standard mode (<code>SPORT_CTL_A.OPMODE = 0</code>), use <code>SPORT_CTL_A.SLEN</code> of 3 to 31 bits.</p> <p>For I²S / packed / left-justified mode (<code>SPORT_CTL_A.OPMODE = 1</code>), use <code>SPORT_CTL_A.SLEN</code> of 4 to 31 bits.</p>
3 (R/W)	LSBF	<p>Least-Significant Bit First.</p> <p>The <code>SPORT_CTL_A.LSBF</code> bit selects whether the half SPORT transmits or receives data LSB first or MSB first.</p>
		0 MSB first sent/received (big endian)
		1 LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	<p>Data Type.</p> <p>The <code>SPORT_CTL_A.DTYPE</code> bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (<code>SPORT_CTL_A.OPMODE = 0</code>).</p>
		0 Right-justify data, zero-fill unused MSBs
		1 Right-justify data, sign-extend unused MSBs
		2 u-law compand data
		3 A-law compand data
0 (R/W)	SPENPRI	<p>Serial Port Enable (Primary).</p> <p>The <code>SPORT_CTL_A.SPENPRI</code> bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.</p>
		0 Disable
		1 Enable

Half SPORT 'B' Control Register

The `SPORT_CTL_B` register contains transmit and receive control bits for SPORT half 'B', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the `SPORT_CTL_B` register vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

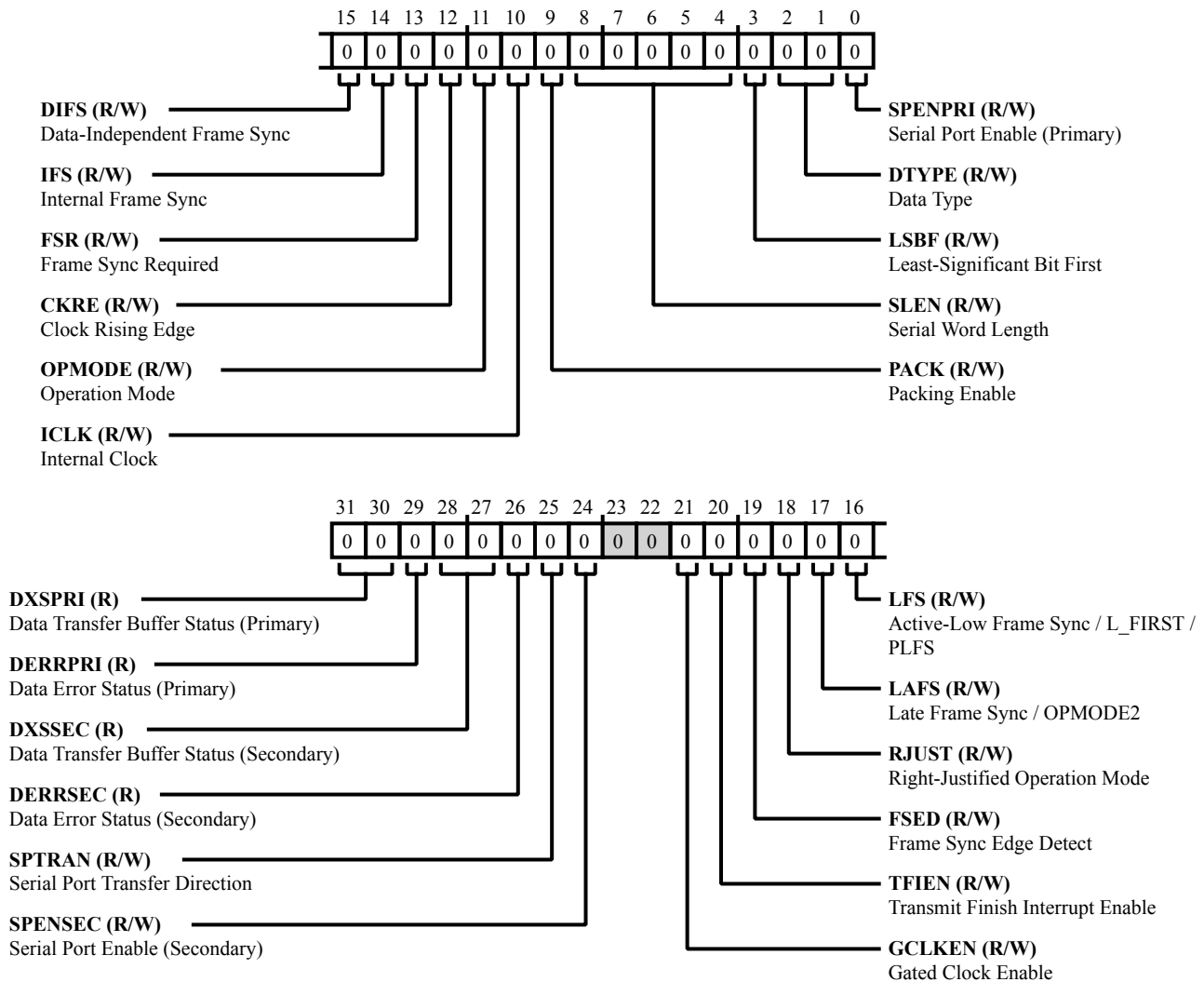


Figure 23-25: `SPORT_CTL_B` Register Diagram

Table 23-26: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The <code>SPORT_CTL_B.DXSPRI</code> bit field indicates the status of the half SPORT's primary channel data buffer.
		0 Empty
		1 Reserved
		2 Partially full
		3 Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The <code>SPORT_CTL_B.DERRPRI</code> bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the <code>SPORT_CTL_B.FSR</code> bit =1, the <code>SPORT_CTL_B.DERRPRI</code> bit indicates whether the <code>SPORT_BFS</code> signal (from an internal or external source) occurred while the <code>SPORT_TXPRI_B</code> data buffer was empty (during transmit) or the <code>SPORT_RXPRI_B</code> data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the <code>SPORT_BFS</code> signal. It is important to note that, as a receiver, the <code>SPORT_CTL_B.DERRPRI</code> bit indicates when the channel has received new data while the <code>SPORT_RXPRI_B</code> receive buffer is full. This new data overwrites existing data. If the <code>SPORT_CTL_B.FSR</code> bit =0, the <code>SPORT_CTL_B.DERRPRI</code> bit is set whenever the SPORT is required to transmit while the <code>SPORT_TXPRI_B</code> transmit buffer is empty and is set whenever the SPORT is required to receive while the <code>SPORT_RXPRI_B</code> receive buffer is full. The SPORT clears the <code>SPORT_CTL_B.DERRPRI</code> bit if the <code>SPORT_ERR_B.DERRPSTAT</code> bit is cleared.
		0 No error
		1 Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The <code>SPORT_CTL_B.DXSSEC</code> bit field indicates the status of the half SPORT's secondary channel data buffer.
		0 Empty
		1 Reserved
		2 Partially full
		3 Full

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_B.DERRSEC bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, the SPORT_CTL_B.DERRSEC bit indicates whether the SPORT_BFS signal (from an internal or external source) occurred while the SPORT_TXSEC_B data buffer was empty (during transmit) or the SPORT_RXSEC_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRSEC bit indicates when the channel has received new data while the SPORT_RXSEC_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, the SPORT_CTL_B.DERRSEC bit is set whenever the SPORT is required to transmit while the SPORT_TXSEC_B transmit buffer is empty. It is also set whenever the SPORT is required to receive while the SPORT_RXSEC_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRSEC bit if the SPORT_ERR_B.DERRSSTAT bit is cleared.
		0 No error
		1 Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The SPORT_CTL_B.SPTRAN bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the SPORT_BCLK and SPORT_BFS pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the SPORT_BCLK and SPORT_BFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.
		0 Receive
		1 Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_B.SPENSEC bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.
		0 Disable
		1 Enable

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	GCLKEN	Gated Clock Enable. The <code>SPORT_CTL_B.GCLKEN</code> bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (<code>SPORT_CTL_B.OPMODE = 0</code> or <code>1</code>). This bit is ignored when the half SPORT is in right-justified mode (<code>SPORT_CTL_B.RJUST = 1</code>) or multichannel mode (<code>SPORT_MCTL_B.MCE = 1</code>). When <code>SPORT_CTL_B.GCLKEN</code> is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).
		0 Disable
		1 Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The <code>SPORT_CTL_B.TFIEN</code> bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the <code>DMA_CFG.INT</code> configuration. When enabled (<code>SPORT_CTL_B.TFIEN = 1</code>), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (<code>SPORT_CTL_B.TFIEN = 0</code>), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).
		0 Last word sent (DMA count done) interrupt
		1 Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The <code>SPORT_CTL_B.FSED</code> bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The <code>SPORT_CTL_B.FSED</code> may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (<code>SPORT_CTL_B.FSED = 0</code>), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.
		0 Level detect frame sync
		1 Edge detect frame sync

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	RJUST	Right-Justified Operation Mode. The <code>SPORT_CTL_B.RJUST</code> bit enables the half SPORT (if <code>SPORT_CTL_B.OPMODE = 1</code>) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the <code>SPORT_MCTL_B.WOFFSET</code> field). For information about appropriate delay selections, see the SPORT operating modes section.
		0 Disable
		1 Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (<code>SPORT_CTL_B.OPMODE = 0</code>) or in right-justified mode (<code>SPORT_CTL_B.RJUST = 1</code>), the <code>SPORT_CTL_B.LAFS</code> bit selects whether the half SPORT generates a late frame sync (<code>SPORT_BFS</code> during first data bit) or generates an early frame sync signal (<code>SPORT_BFS</code> during serial clock cycle before first data bit). When the half SPORT is in I ² S / left-justified mode (<code>SPORT_CTL_B.OPMODE = 1</code>), the <code>SPORT_CTL_B.LAFS</code> bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I ² S mode. When the half SPORT is in multichannel mode (<code>SPORT_MCTL_B.MCE = 1</code>), the <code>SPORT_CTL_B.LAFS</code> bit is reserved.
		0 Early frame sync (or I ² S mode)
		1 Late frame sync (or left-justified mode)

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I ² S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LFS bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.
		0 Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I ² S/packed mode) or left channel first (left-justified mode)
		1 Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I ² S/packed mode) or right channel first (left-justified mode)
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_B.DIFS bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by SPORT_DIV_B.FSDIV. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the SPORT_CTL_B.DIFS bit is automatically set when the half SPORT is in packed or multichannel modes.
		0 Data-dependent frame sync
		1 Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The SPORT_CTL_B.IFS bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.
		0 External frame sync
		1 Internal frame sync

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_B.FSR selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I ² S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1) or is in multichannel mode (SPORT_MCTL_B.MCE =1).
		0 No frame sync required
		1 Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_B.CKRE selects the rising or falling edge of the SPORT_BCLK clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the SPORT_BCLK. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for SPORT_CTL_B.CKRE. This programming drives the internally-generated signals on one edge of SPORT_BCLK and samples the received signals on the opposite edge.
		0 Clock falling edge
		1 Clock rising edge
11 (R/W)	OPMODE	Operation Mode. The SPORT_CTL_B.OPMODE bit selects whether the half SPORT operates in DSP standard / multichannel mode or operates in I ² S / packed / left-justified mode. The mode selection affects the operation of the SPORT_CTL_B.LAFS and SPORT_CTL_B.LFS bits. Also, the SPORT_CTL_B.OPMODE bit enables or disables operation of the SPORT_CTL_B.GCLKEN, SPORT_CTL_B.FSED, SPORT_CTL_B.RJUST, SPORT_CTL_B.DIFS, SPORT_CTL_B.FSR, and SPORT_CTL_B.CKRE bits.
		0 DSP standard/multichannel mode
		1 I ² S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_BCLK clock signal, and the SPORT_BCLK is an output. The SPORT_DIV_B.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_BCLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.
		0 External clock
		1 Internal clock

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	PACK	Packing Enable. The <code>SPORT_CTL_B.PACK</code> bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.
		0 Disable
		1 Enable
8:4 (R/W)	SLEN	Serial Word Length. The <code>SPORT_CTL_B.SLEN</code> bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: $SPORT_CTL_B.SLEN = (\text{serial word length in bits}) - 1$ For DSP standard mode (<code>SPORT_CTL_B.OPMODE = 0</code>), use <code>SPORT_CTL_B.SLEN</code> of 3 to 31 bits. For I ² S / packed / left-justified mode (<code>SPORT_CTL_B.OPMODE = 1</code>), use <code>SPORT_CTL_B.SLEN</code> of 4 to 31 bits.
3 (R/W)	LSBF	Least-Significant Bit First. The <code>SPORT_CTL_B.LSBF</code> bit selects whether the half SPORT transmits or receives data LSB first or MSB first.
		0 MSB first sent/received (big endian)
		1 LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The <code>SPORT_CTL_B.DTYPE</code> bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (<code>SPORT_CTL_B.OPMODE = 0</code>).
		0 Right-justify data, zero-fill unused MSBs
		1 Right-justify data, sign-extend unused MSBs
		2 u-law compand data
		3 A-law compand data

Table 23-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The SPORT_CTL_B.SPENPRI bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'A' Divisor Register

The `SPORT_DIV_A` register contains divisor values that determine frequencies of internally-generated clocks and frame syncs for half SPORT 'A'.

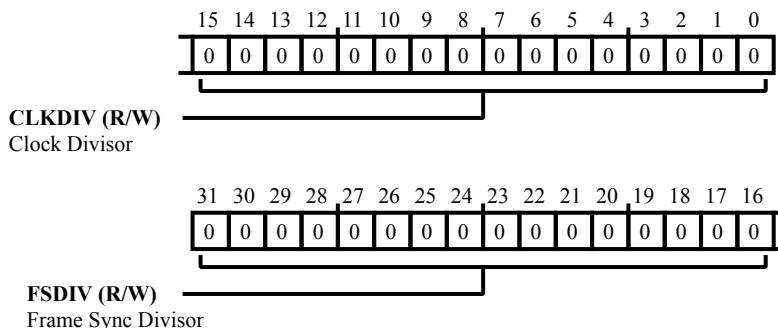


Figure 23-26: SPORT_DIV_A Register Diagram

Table 23-27: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	<p>Frame Sync Divisor.</p> <p>The <code>SPORT_DIV_A.FSDIV</code> bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating <code>SPORT_DIV_A.FSDIV</code> to the number of cycles between frame sync pulses is:</p> $\text{SPORT_DIV_A.FSDIV} = (\text{number of serial clocks between frame syncs}) - 1$ <p>Use the following equation to determine the value of <code>SPORT_DIV_A.FSDIV</code>, given the serial clock frequency and desired frame sync frequency:</p> $\text{FSDIV} = (\text{SPORT_ACLK} / \text{SPORT_AFS}) - 1$ <p>Note that the frame sync is continuously active when <code>SPORT_DIV_A.FSDIV = 0</code>. The value of <code>SPORT_DIV_A.FSDIV</code> should not be less than the serial word length (<code>SPORT_CTL_A.SLEN</code>), as this may cause an external device to abort the current operation or cause other unpredictable results.</p>
15:0 (R/W)	CLKDIV	<p>Clock Divisor.</p> <p>The <code>SPORT_DIV_A.CLKDIV</code> bits select the divisor that the half SPORT uses to calculate the serial clock (<code>SPORT_ACLK</code>) from the processor system clock (<code>SCLK</code>). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (<code>SPORT_CTL_A.ICLK = 1</code>), legal <code>SPORT_DIV_A.CLKDIV</code> values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of <code>SPORT_DIV_A.CLKDIV</code>:</p> $\text{CLKDIV} = (\text{SCLK} / \text{SPORT_ACLK}) - 1$ <p>For the maximum serial clock frequency, see the processor data sheet.</p>

Half SPORT 'B' Divisor Register

The `SPORT_DIV_B` contains divisor values that determine frequencies of internally-generated clocks and frame syncs for SPORT half 'B'.

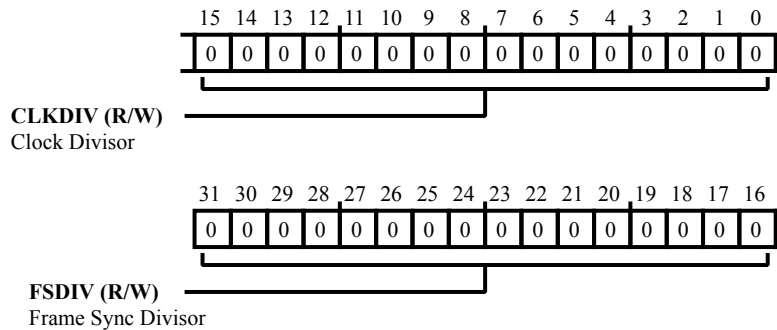


Figure 23-27: `SPORT_DIV_B` Register Diagram

Table 23-28: `SPORT_DIV_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	<p>Frame Sync Divisor.</p> <p>The <code>SPORT_DIV_B.FSDIV</code> bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating <code>SPORT_DIV_B.FSDIV</code> to the number of cycles between frame sync pulses is:</p> $\text{SPORT_DIV_B.FSDIV} = (\text{number of serial clocks between frame syncs}) - 1$ <p>Use the following equation to determine the value of <code>SPORT_DIV_B.FSDIV</code>, given the serial clock frequency and desired frame sync frequency:</p> $\text{FSDIV} = (\text{SPORT_BCLK} / \text{SPORT_BFS}) - 1$ <p>Note that the frame sync is continuously active when <code>SPORT_DIV_B.FSDIV = 0</code>. The value of <code>SPORT_DIV_B.FSDIV</code> should not be less than the serial word length (<code>SPORT_CTL_B.SLEN</code>), as this may cause an external device to abort the current operation or cause other unpredictable results.</p>
15:0 (R/W)	CLKDIV	<p>Clock Divisor.</p> <p>The <code>SPORT_DIV_B.CLKDIV</code> bits select the divisor that the half SPORT uses to calculate the serial clock (<code>SPORT_BCLK</code>) from the processor system clock (<code>SCLK</code>). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (<code>SPORT_CTL_B.ICLK = 1</code>), legal <code>SPORT_DIV_B.CLKDIV</code> values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of <code>SPORT_DIV_B.CLKDIV</code>:</p> $\text{CLKDIV} = (\text{SCLK} / \text{SPORT_BCLK}) - 1$ <p>For the maximum serial clock frequency, see the processor data sheet.</p>

Half SPORT 'A' Error Register

The `SPORT_ERR_A` register contains error status and error interrupt mask bits for SPORT half 'A', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

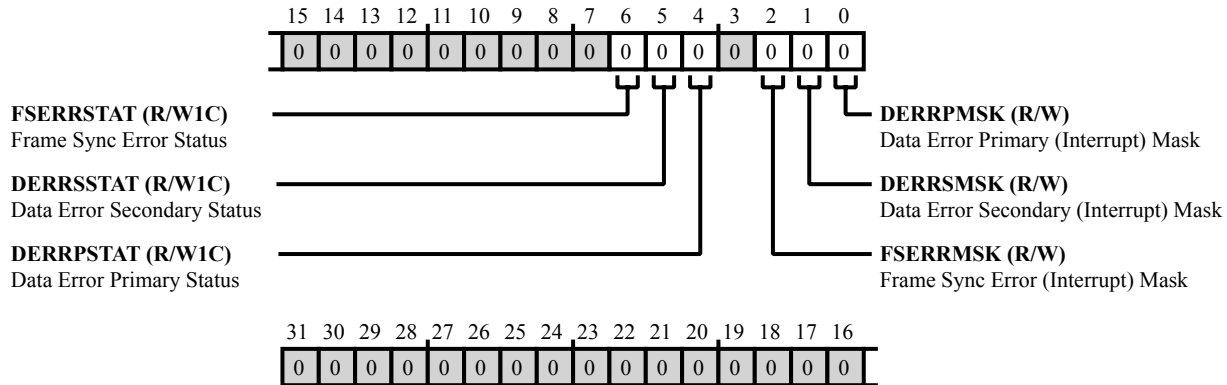


Figure 23-28: `SPORT_ERR_A` Register Diagram

Table 23-29: `SPORT_ERR_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	FSERRSTAT	Frame Sync Error Status. The <code>SPORT_ERR_A.FSERRSTAT</code> bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, <code>SPORT_CTL_A.SLEN = 31</code>). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.
		0 No error
		1 Error (non-zero bit count at frame sync)

Table 23-29: SPORT_ERR_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_A.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), the SPORT_ERR_A.DERRSSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), the SPORT_ERR_A.DERRSSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRSEC status bit.
		0 No error
		1 Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_A.DERPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), the SPORT_ERR_A.DERPSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), the SPORT_ERR_A.DERPSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only SPORT_CTL_A.DERPRI status bit.
		0 No error
		1 Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_A.FSERRMSK unmask (enables) the half SPORT to generate the frame sync error interrupt request.
		0 Mask (disable)
		1 Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_A.DERRSMSK unmask (enables) the half SPORT to generate the data error interrupt request for the secondary channel.
		0 Mask (disable)
		1 Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_A.DERRPMSK unmask (enables) the half SPORT to generate the data error interrupt request for the primary channel.
		0 Mask (disable)
		1 Unmask (enable)

Half SPORT 'B' Error Register

The `SPORT_ERR_B` register contains error status and error interrupt mask bits for SPORT half 'B', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

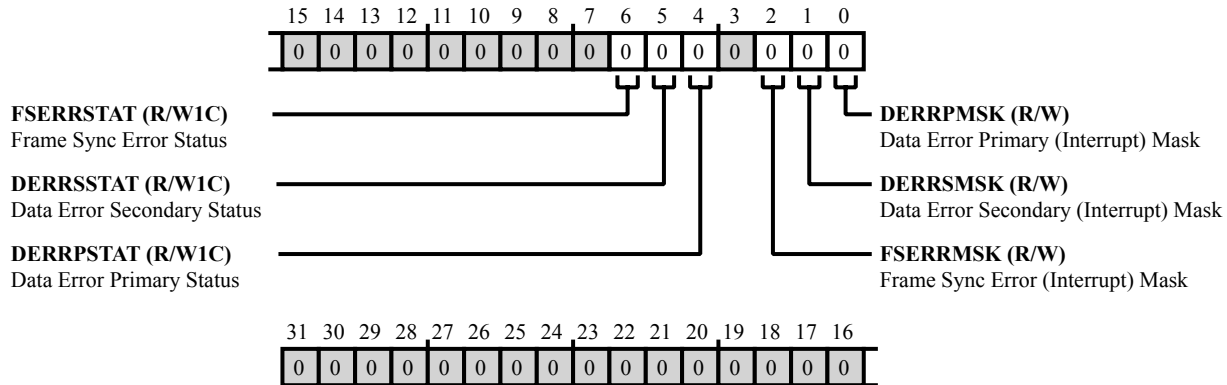


Figure 23-29: `SPORT_ERR_B` Register Diagram

Table 23-30: `SPORT_ERR_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	FSERRSTAT	<p>Frame Sync Error Status.</p> <p>The <code>SPORT_ERR_B.FSERRSTAT</code> bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, <code>SPORT_CTL_B.SLEN = 31</code>). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.</p>
		0 No error
		1 Error (non-zero bit count at frame sync)

Table 23-30: SPORT_ERR_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_B.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), SPORT_ERR_B.DERRSSTAT indicates the transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRSSTAT indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRSEC status bit.
		0 No error
		1 Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_B.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), the SPORT_ERR_B.DERRPSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), the SPORT_ERR_B.DERRPSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRPRI status bit.
		0 No error
		1 Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_B.FSERRMSK unmask (enables) the half SPORT to generate the frame sync error interrupt request.
		0 Mask (disable)
		1 Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_B.DERRSMSK unmask (enables) the half SPORT to generate the data error interrupt request for the secondary channel.
		0 Mask (disable)
		1 Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_B.DERRPMSK unmask (enables) the half SPORT to generate the data error interrupt request for the primary channel.
		0 Mask (disable)
		1 Unmask (enable)

Half SPORT 'A' Multichannel Control Register

The `SPORT_MCTL_A` register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

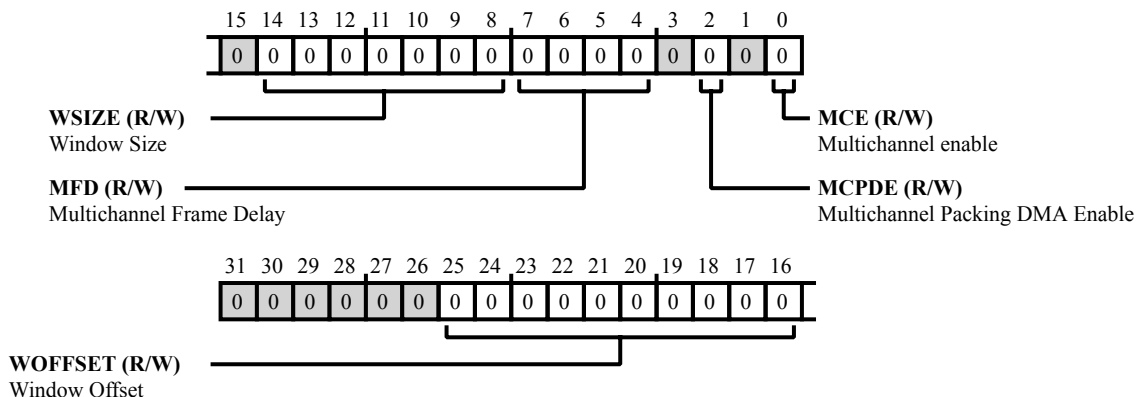


Figure 23-30: `SPORT_MCTL_A` Register Diagram

Table 23-31: `SPORT_MCTL_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The <code>SPORT_MCTL_A.WOFFSET</code> bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (<code>SPORT_MCTL_A.MCE = 0</code>) and the right-justified mode is enabled (<code>SPORT_CTL_A.RJUST = 1</code>), the least significant 6 bits of <code>SPORT_MCTL_A.WOFFSET</code> serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The <code>SPORT_MCTL_A.WSIZE</code> bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: $SPORT_MCTL_A.WSIZE = (\text{number of channel slots}) - 1$
7:4 (R/W)	MFD	Multichannel Frame Delay. The <code>SPORT_MCTL_A.MFD</code> bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting multichannel frame delay of 0-15 serial clocks.

Table 23-31: SPORT_MCTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The <code>SPORT_MCTL_A.MCPDE</code> bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.
		0 Disable
		1 Enable
0 (R/W)	MCE	Multichannel enable. The <code>SPORT_MCTL_A.MCE</code> bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if <code>SPORT_CTL_A.OPMODE=0</code> ; while it is configured in packed mode if <code>SPORT_CTL_A.OPMODE=1</code> . When configuring in these modes, the multichannel enable bit (<code>SPORT_MCTL_A.MCE</code>) should be set before enabling the SPORT data channel enable bits (<code>SPORT_CTL_A.SPENPRI</code> and/or <code>SPORT_CTL_A.SPENSEC</code>). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the <code>SPORT_CTL_A.DERRPRI</code> and <code>SPORT_CTL_A.DERRSEC</code> bits are cleared.
		0 Disable
		1 Enable

Half SPORT 'B' Multichannel Control Register

The `SPORT_MCTL_B` register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

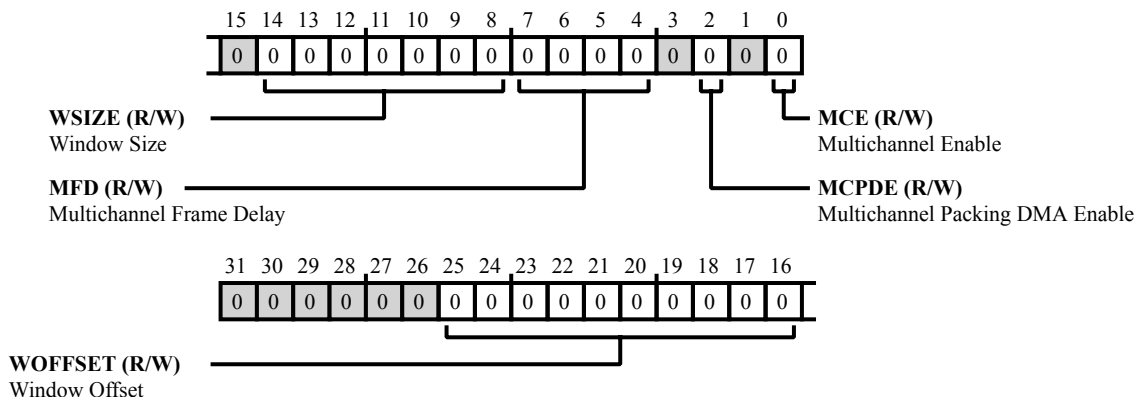


Figure 23-31: `SPORT_MCTL_B` Register Diagram

Table 23-32: `SPORT_MCTL_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The <code>SPORT_MCTL_B.WOFFSET</code> bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (<code>SPORT_MCTL_B.MCE = 0</code>) and right-justified mode is enabled (<code>SPORT_CTL_B.RJUST = 1</code>), the least significant 6 bits of <code>SPORT_MCTL_B.WOFFSET</code> serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The <code>SPORT_MCTL_B.WSIZE</code> bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: $SPORT_MCTL_B.WSIZE = (\text{number of channel slots}) - 1$
7:4 (R/W)	MFD	Multichannel Frame Delay. The <code>SPORT_MCTL_B.MFD</code> bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting a multichannel frame delay of 0-15 serial clocks.

Table 23-32: SPORT_MCTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The SPORT_MCTL_B.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.
		0 Disable
		1 Enable
0 (R/W)	MCE	Multichannel Enable. The SPORT_MCTL_B.MCE bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if SPORT_CTL_B.OPMODE=0; while it is configured in packed mode if SPORT_CTL_B.OPMODE=1. When configuring in these modes, the multichannel enable bit (SPORT_MCTL_B.MCE) should be set before enabling SPORT data channel enable bits (SPORT_CTL_B.SPENPRI and/or SPORT_CTL_B.SPENSEC). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the SPORT_CTL_B.DERRPRI and SPORT_CTL_B.DERRSEC bits are cleared.
		0 Disable
		1 Enable

Half SPORT 'A' Multichannel Status Register

The `SPORT_MSTAT_A` register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the `SPORT_MSTAT_A` register restarts at 0 at each frame sync.

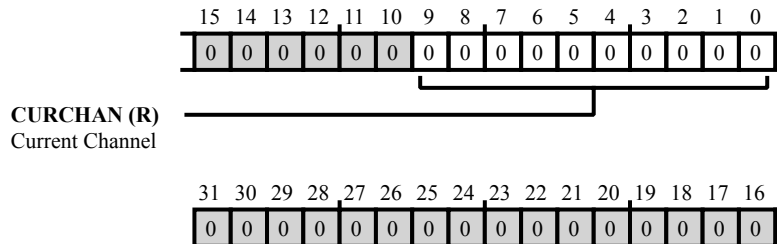


Figure 23-32: `SPORT_MSTAT_A` Register Diagram

Table 23-33: `SPORT_MSTAT_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The <code>SPORT_MSTAT_A.CURCHAN</code> bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'B' Multichannel Status Register

The `SPORT_MSTAT_B` register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the `SPORT_MSTAT_B` register restarts at 0 at each frame sync.

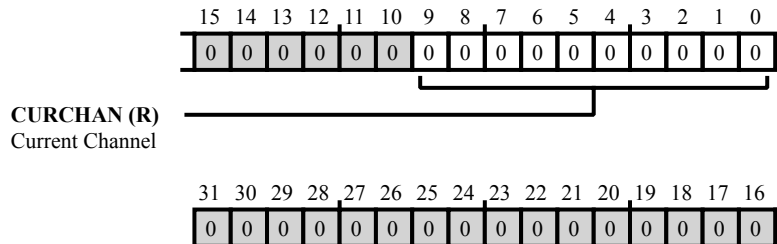


Figure 23-33: `SPORT_MSTAT_B` Register Diagram

Table 23-34: `SPORT_MSTAT_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The <code>SPORT_MSTAT_B.CURCHAN</code> bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'A' Rx Buffer (Primary) Register

The `SPORT_RXPRI_A` register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXPRI_A` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXPRI_A` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

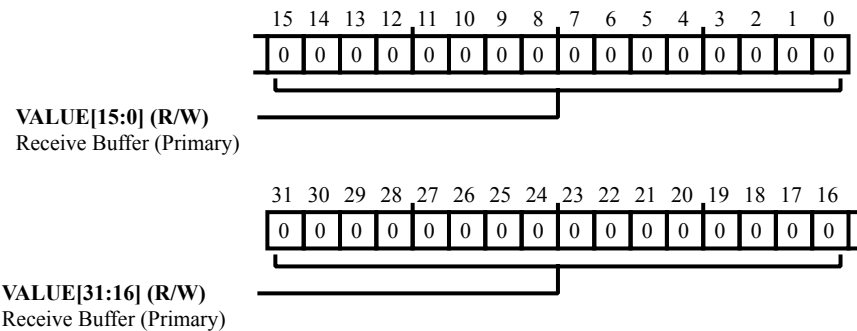


Figure 23-34: `SPORT_RXPRI_A` Register Diagram

Table 23-35: `SPORT_RXPRI_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The <code>SPORT_RXPRI_A.VALUE</code> bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_A.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_A</code> and <code>SPORT_MCTL_A</code> register descriptions.

Half SPORT 'B' Rx Buffer (Primary) Register

The `SPORT_RXPRI_B` register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXPRI_B` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXPRI_B` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

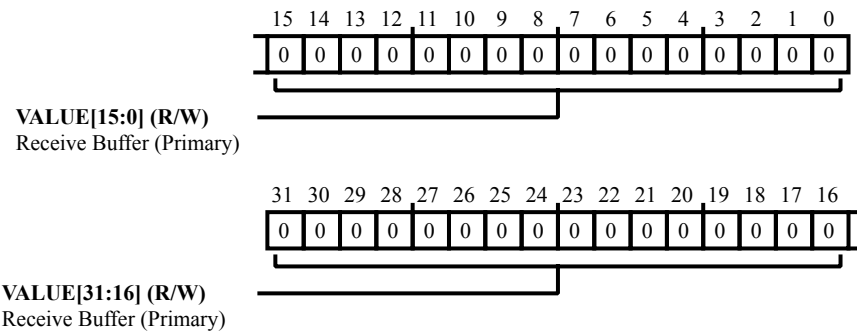


Figure 23-35: `SPORT_RXPRI_B` Register Diagram

Table 23-36: `SPORT_RXPRI_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The <code>SPORT_RXPRI_B.VALUE</code> bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_B.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_B</code> and <code>SPORT_MCTL_B</code> register descriptions.

Half SPORT 'A' Rx Buffer (Secondary) Register

The `SPORT_RXSEC_A` register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXSEC_A` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXSEC_A` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

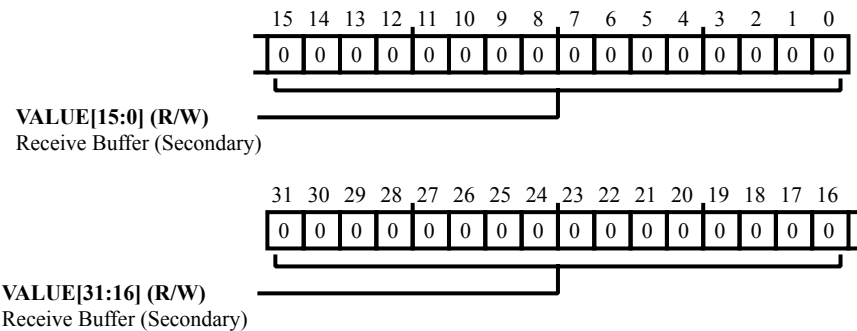


Figure 23-36: `SPORT_RXSEC_A` Register Diagram

Table 23-37: `SPORT_RXSEC_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The <code>SPORT_RXSEC_A.VALUE</code> bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_A.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_A</code> and <code>SPORT_MCTL_A</code> register descriptions.

Half SPORT 'B' Rx Buffer (Secondary) Register

The `SPORT_RXSEC_B` register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXSEC_B` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXSEC_B` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

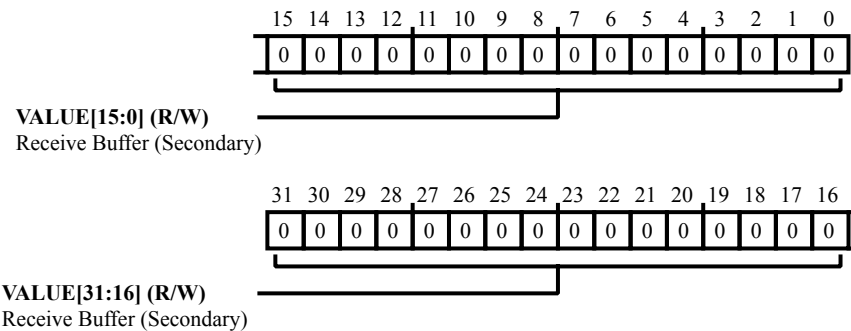


Figure 23-37: `SPORT_RXSEC_B` Register Diagram

Table 23-38: `SPORT_RXSEC_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The <code>SPORT_RXSEC_B.VALUE</code> bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_B.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_B</code> and <code>SPORT_MCTL_B</code> register descriptions.

Half SPORT 'A' Tx Buffer (Primary) Register

The `SPORT_TXPRI_A` register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXPRI_A` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_A.PACK = 0`); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_A.PACK = 1`). So, depending on the `PACK` bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXPRI_A` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

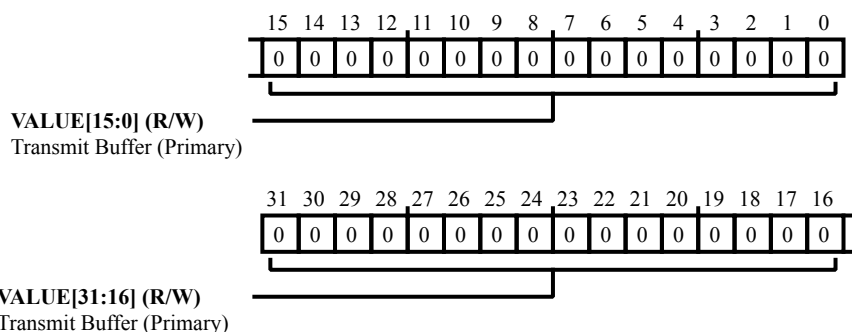


Figure 23-38: `SPORT_TXPRI_A` Register Diagram

Table 23-39: `SPORT_TXPRI_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The <code>SPORT_TXPRI_A.VALUE</code> bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_A.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_A</code> and <code>SPORT_MCTL_A</code> register descriptions.

Half SPORT 'B' Tx Buffer (Primary) Register

The `SPORT_TXPRI_B` register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXPRI_B` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_B.PACK = 0`); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_B.PACK = 1`). So, depending on the `PACK` bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXPRI_B` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

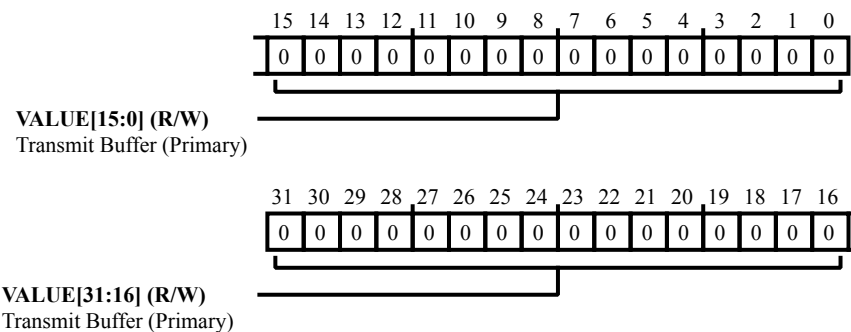


Figure 23-39: `SPORT_TXPRI_B` Register Diagram

Table 23-40: `SPORT_TXPRI_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The <code>SPORT_TXPRI_B.VALUE</code> bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_B.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_B</code> and <code>SPORT_MCTL_B</code> register descriptions.

Half SPORT 'A' Tx Buffer (Secondary) Register

The `SPORT_TXSEC_A` register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXSEC_A` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_A.PACK = 0`); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_A.PACK = 1`). So, depending on the `PACK` bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXSEC_A` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

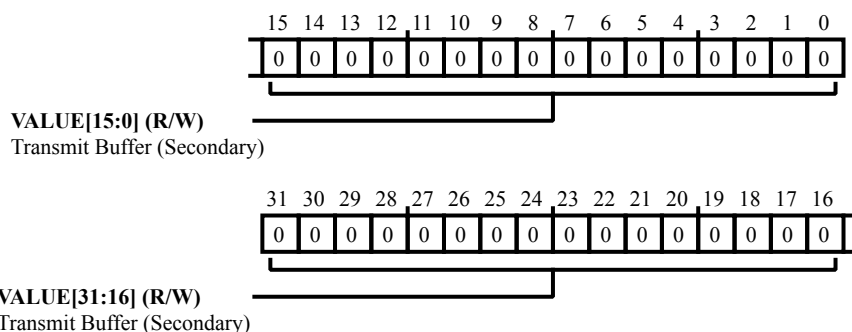


Figure 23-40: `SPORT_TXSEC_A` Register Diagram

Table 23-41: `SPORT_TXSEC_A` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The <code>SPORT_TXSEC_A.VALUE</code> bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_A.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_A</code> and <code>SPORT_MCTL_A</code> register descriptions.

Half SPORT 'B' Tx Buffer (Secondary) Register

The `SPORT_TXSEC_B` register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXSEC_B` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_B.PACK = 0`); while it acts as two-location buffer when packing is enabled (`SPORT_CTL_B.PACK = 1`). So, depending on the `PACK` bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXSEC_B` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

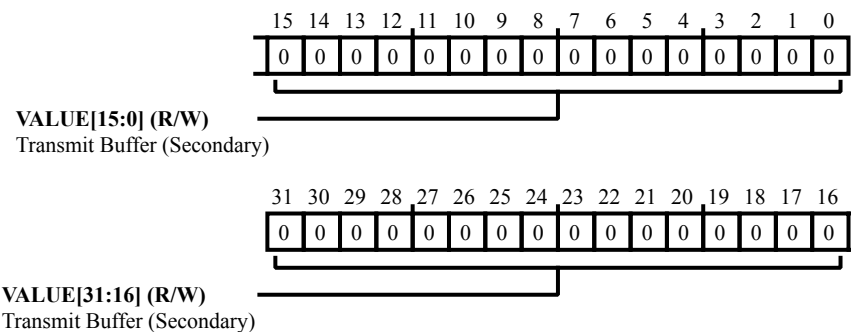


Figure 23-41: `SPORT_TXSEC_B` Register Diagram

Table 23-42: `SPORT_TXSEC_B` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The <code>SPORT_TXSEC_B.VALUE</code> bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the <code>SPORT_MCTL_B.MCE</code>) empty the contents of this data buffer. For more information, see the <code>SPORT_CTL_B</code> and <code>SPORT_MCTL_B</code> register descriptions.

24 Precision Clock Generators (PCG)

The precision clock generators are used to produce a pair of signals from a clock input signal. The two signals generated are normally used as a serial bit clock and frame sync pair. The PCG is part of the DAI. There are two PCG signals in each DAI: *PCGA* and *PCGB* belong to DAI0, *PCGC* and *PCGD* belong to DAI1.

Features

The following list describes the features of the precision clock generators.

- SRU allows the routing of all of the PCG signals in one DAI (four PCG signals in DAI0 or DAI1)
- Input clock selection: *SYS_CLKIN0* can be used as the input clock for PCG, when configured to use *CLKIN* as clock source.
- Provides four different clock dividers for serial clock, frame sync, phase (20-bit), and pulse width (16-bit)
- Phase shift allows adjustment of the frame sync relative to the serial clock and can be shifted the full period and wrapped around
- Provides pulse width control for arbitrary frame sync signal generation
- Bypass mode for external frame sync manipulation
- External trigger mode starts PCG operation
- No additional jitter is introduced when using off-chip clocks

Functional Description

The *PCG Block Diagram* shows the blocks within the module and its connection to the DAI. The following sections provide information on the function of these blocks.

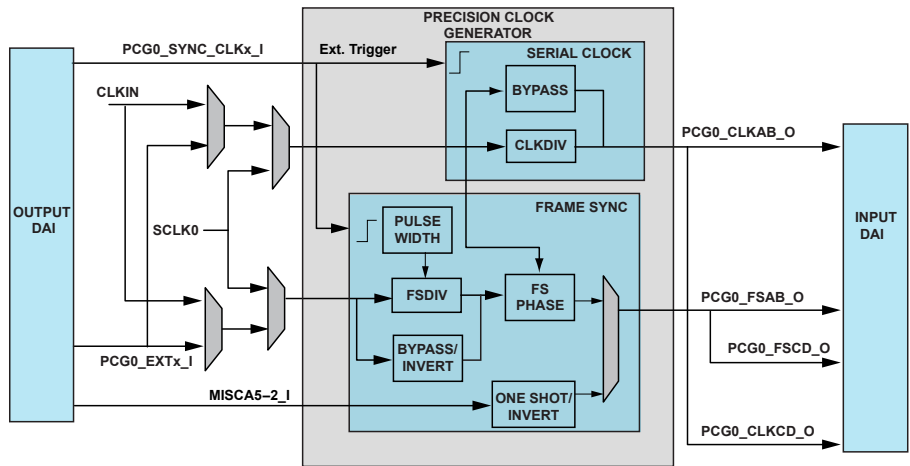


Figure 24-1: PCG Block Diagram

ADSP-2156x PCG Register List

Precision Clock Generator

Table 24-1: ADSP-2156x PCG Register List

Name	Description
PCG_CTLA0	Precision Clock A Control 0 Register
PCG_CTLA1	Precision Clock A Control 1 Register
PCG_CTLB0	Precision Clock B Control 0 Register
PCG_CTLB1	Precision Clock B Control 1 Register
PCG_CTLC0	Precision Clock C Control 0 Register
PCG_CTLC1	Precision Clock C Control 1 Register
PCG_CTLD0	Precision Clock D Control 0 Register
PCG_CTLD1	Precision Clock D Control 1 Register
PCG_PW1	Precision Clock Pulse Width Control 1 Register
PCG_PW2	Precision Clock Pulse Width Control 2 Register
PCG_SYNC1	Precision Clock Frame Sync Synchronization 1 Register
PCG_SYNC2	Precision Clock Frame Sync Synchronization 2 Register

Internal Interface

The fundamental clock of the PCG is *SCLK0*. The clock to this module can be shut off for power savings.

Serial Clock

Each of the four units (A, B, C, and D) produces a clock output. Serial clock generation from a unit is independently enabled and controlled. Sources for the serial clock generation can be either from the CLKIN, SCLK0, or a DAI pin source.

When CLKIN is chosen as input clock in PCG, the clock source is SYS_CLKIN0.

Note that the divider is working in normal mode for $PCG0_CTLx1.CLKDIV > 1$. For $PCG_CTLA1.CLKDIV/PCG_CTLB1.CLKDIV = 0$ or 1, the divider operates in bypass mode (input clock is fed directly to its output). In bypass mode, the clock at the output can theoretically run at up to the SCLK0 frequency. Check the data sheet for the specified maximum operation speed of the DAI pin buffers.

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.

NOTE: A PCG clock output cannot be fed to its own input.

Frame Sync

The following sections describe the use of frame syncs in the PCGs.

Frame Sync Output

Each of the four units (A through D) produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they must accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, SCLK0, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit.

If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, a 20-bit divisor controls frame sync generation.

Divider Mode Selection

If a frame sync divisor is greater than 1, the PCG frame sync output frequency is equal to the input clock frequency, divided by a 20-bit integer. This integer is specified in the $PCG_CTLA0.FSDIV/PCG_CTLB0.FSDIV$ bit field (bits 19:0).

However, if the frame sync divisor is 0 or 1, the PCG's frame sync clock generation unit is bypassed, and the frame sync input is connected directly to the frame sync output. For $PCG_CTLB0.FSDIV = 0, 1$ the [PCG_PW1/PCG_PW2](#) registers function differently than in normal mode.

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync relative to the input clock of the same unit. This feature allows the shifting of the frame sync signal in time relative to the clock input signal. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

For example, the I²S protocol specifies that the frame sync transition from high-to-low occurs one clock cycle before the beginning of a frame. Since an I²S frame is 64 clock cycles long, delaying the frame sync by 63 cycles produces the required framing.

Phase shifting is represented as a full 20-bit value. Even when the frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.

NOTE: Phase shifting is specified as a 2 x 10-bit divider value in the `PCG_CTLA0.FSPHASEHI` bit field (bits 29:20) and in the `PCG_CTLA1.FSPHASELO` bit field (bits 29:20).

A single 20-bit value spans these two-bit fields. The upper half of the word (bits 19:10) is in the `PCG_CTLA0` register, and the lower half (bits 9:0) is in the `PCG_CTLA1` register.

The phase shift between clock and frame sync outputs can be programmed using the `PCG_PW1/PCG_PW2` registers and all of the control registers under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- The clock and frame sync are enabled at the same time using a single atomic instruction.
- The frame sync divisor is an integral multiple of the clock divisor.

NOTE: When using a clock and frame sync as a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction.

NOTE: If the phase shift is 0 (see the *Phase and Pulse Width Settings* figure), the clock and frame sync outputs rise at the same time.

If the phase shift is 1, the frame sync output transitions one input clock period ahead of the clock transition.

If the phase shift is *divisor* – 1, the frame sync transitions *divisor* – 1 input clock periods ahead of the clock transitions.

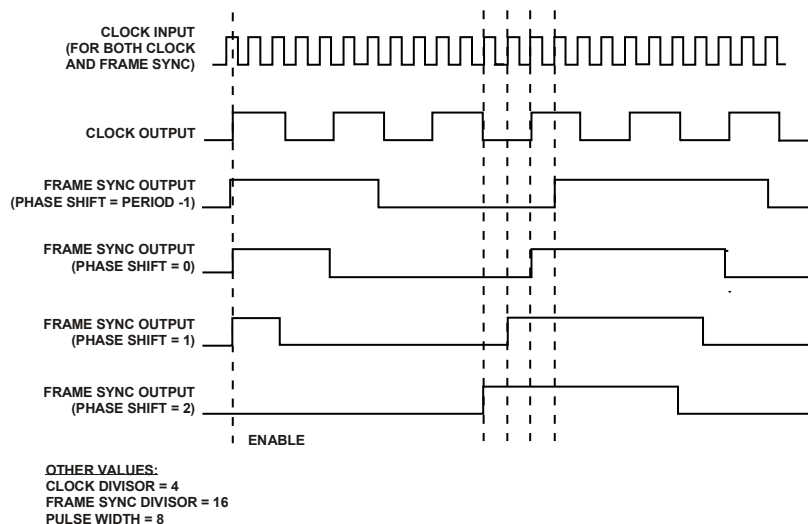


Figure 24-2: Phase and Pulse Width Settings

NOTE: When generating single frame sync pulses (the length of one SPORT clock cycle), take care with respect to the drive and sampling edges. If the rules are violated, for example if the SPORT is not driving data, the module cannot detect a valid sample edge.

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high.

A 16-bit value determines the width of the framing pulse. Settings for pulse width can range from zero to $DIV - 1$. The pulse width should be less than the divisor of the frame sync. The pulse width of frame sync is specified in the `PCG_PW1.FSA`, `PCG_PW1.FSB`, `PCG_PW2.FSC`, and `PCG_PW2.FSD` bits (15-0) and (31-16).

Default Pulse Width

If the pulse width count is equal to 0 and if the `PCG_CTLA0.FSDIV/PCG_CTLB0.FSDIV` bit field is even, then the actual pulse width of the frame sync output is equal to:

For even divisors: frame sync divisor/2

If the pulse width count is equal to 0 and if the `PCG_CTLA0.FSDIV/PCG_CTLB0.FSDIV` bit field is odd, then the actual pulse width of the frame sync output is equal to:

For odd divisors: frame sync divisor $- 1/2$

Input Clock Source Considerations

The core Phase-Locked Loop (PLL) has been designed to provide clocking for the processor core. The performance specifications of this PLL are appropriate for the core. But, they have not been optimized or specified for precision data converters where jitter directly translates into time quantization errors and distortion.

Therefore, the PCG allows the routing of external clock sources which are independent of the core PLL.

Timing Example for I²S Mode

For I²S mode, the frame sync should be driven at the falling edge of SPORT clock. In other words, the frame sync edge must coincide with the falling edge of the SPORT clock. To satisfy this requirement, program the phase of the frame sync accordingly in the PCG control registers.

For example, assume that the input clock source for both clock and frame sync are the same and both the clock and frame sync are enabled at the same time. Also, assume that the clock divisor value for generating the required SPORT clock is `PCG_CTLA1.CLKDIV = 4`. Then, for a 32-bit word length, the frame sync divisor value is:

`PCG_CTLA0.FSDIV = 64, PCG_CTLA1.CLKDIV = 256.`

By default, for phase = 0, the rising edge of both SPORT clock and frame sync coincide. To make sure that the frame sync edges coincide with the falling edge of the SPORT clock, program the phase value as:

`PCG_CTLA1.CLKDIV/2 = 2.`

Cross Mode Connections

The symmetric dual DAI architecture allows cross connections between both PCGs (A,B) and (C,D) to the other DAI. Each PCG (A through D) supports an alternative input clock (`PCG0_EXTx_I`) (see [Figure 24-1 PCG Block Diagram](#)) which can be sourced via a DAI pin buffer from the other DAI. Note however if routing a source (clock or FS) only DAI pin buffer 2 to 20 can be used (DAI pin buffer 1 is no longer available and is replaced by the DAI CRS buffer for the other DAI). See [DAI Routing Capabilities](#) for more information.

Operating Modes

The following sections provide information on the operating modes of the precision clock generator.

Normal Mode

When the frame sync divisor is set to any value other than zero or one, the PCGs operate in normal mode. In normal mode, the divisor determines the frequency of the frame sync output where:

Frequency of Frame Sync Output = Input Frequency/Divisor

The value of the pulse width control determines the high period of the frame sync output. The value of the pulse width control must be less than the value of the divisor.

The value of the phase control determines the phase of the frame sync output. If the phase is zero, then the positive edges of the clock and frame sync coincide when:

- the clock and frame sync dividers are enabled at the same time using an atomic instruction
- the divisors of the clock and frame sync are the same
- the source for the clock and frame sync is the same

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the difference between the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame sync appears

to lead the clock. If the phase is only slightly less than the frame sync divisor, then the frame sync appears to lag the clock. The frame sync phase must not be greater than the divisor.

Bypass Mode

When the frame sync divisor for the frame sync has a value of zero or one, the frame sync is in bypass mode and the `PCG_PW1/PCG_PW2` register has different functionality than in normal mode.

NOTE: In normal mode, bits 15:0 and 31:18 of the `PCG_PW1/PCG_PW2` register are used to program the pulse width count. In bypass mode, bits 15:2 and 31:18 are ignored.

If the `PCG_PW1.STROBEA` field is cleared, then the input is directly passed (see the *Bypass and Inverted Bypass* figure) to the frame sync output either inverted or not inverted, depending on the `PCG_PW1.INVFSA` bits.

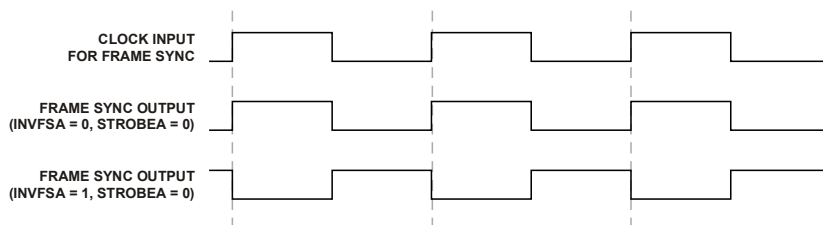


Figure 24-3: Bypass and Inverted Bypass

One-Shot Mode

In the one-shot mode operation shown in the *One Shot Mode PCG A (MISCA2_I Input)* figure, the PCG produces a series of periods but does not run continuously.

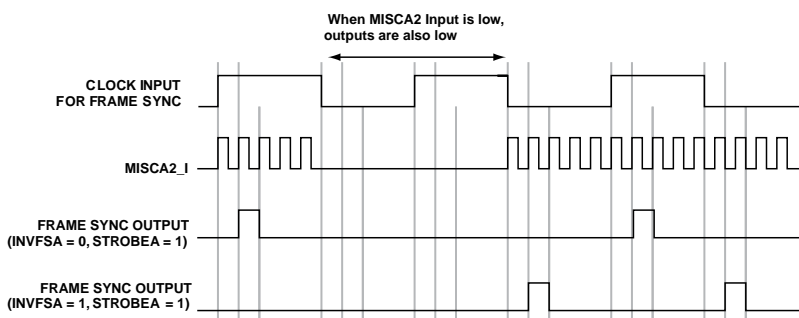


Figure 24-4: One Shot Mode PCG A (MISCA2_I Input)

Bypass mode also enables the generation of a strobe pulse (one-shot frame sync). Strobe usage ignores the divider counters and looks to the SRU to provide the input signal.

The following are the input signals for the respective PCG instances in One-shot mode:

- PCG A: DAI0_MISCA2_I
- PCG B: DAI0_MISCA3_I
- PCG C: DAI1_MISCA2_I

- PCG D: DAI1_MISCA3_I

NOTE: A strobe period is defined to be the period of the FS input clock signal as specified by the `PCG_CTLA1.FSSRC` bit.

Audio System Example

The *PCG Setup for I²S or Left-Justified DAI* figure shows an example of the interconnections between the S/PDIF receiver, ASRC, and the PCGs. The interconnections are made by programming the signal routing unit. It shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (ASRC) to interface to an external audio DAC. The PCG is configured to provide a fixed ASRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player, but can also be from other source at any nominal sample rates.

Similarly, the phase shift for frame syncs B, C, and D is specified in the corresponding control registers (`PCG_CTLA0` through `PCG_CTLB1`).

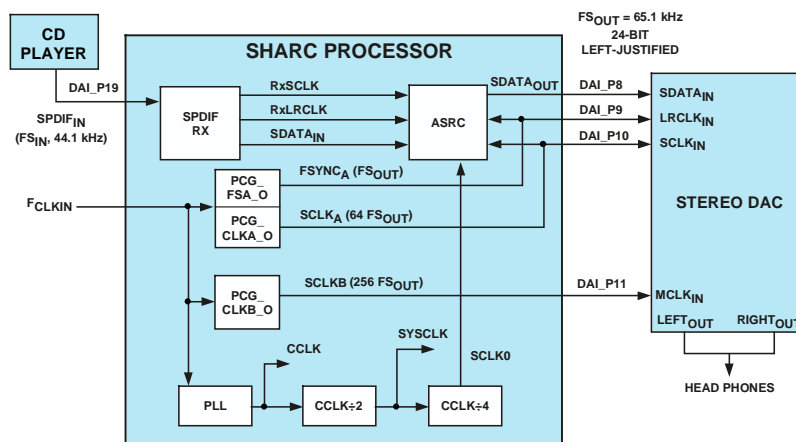


Figure 24-5: PCG Setup for I²S or Left-Justified DAI

Three synchronous clocks are required in audio systems:

1. Frame sync (FS)
2. Serial bit clock (64 FS)
3. Master DAC clock (256 FS)

Since each PCG has only two outputs, this example requires two PCGs. Furthermore, because the digital audio interface requires a fixed-phase relation between serial clock and FS, these two outputs should come from one PCG (PCG A). The master clock comes from the second (PCG B).

The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the ASRCs and external DACs. However, the range of choices is limited by CLKIN and the ratio of `PCG_CLKx_O`: serial clock:FS. The ratio is normally fixed at 256:64:1 to support digital audio left-justified, I²S and right-justified interface modes.

Many DACs also support 384, 512, and 786x FS for `PCG_CLKx_O`, which allows some additional flexibility in choosing serial clock.

Note that the falling edge of serial clock must always be synchronous with both edges of FS. This condition requires that the phase of the serial clock and FS signals for a common PCG (PCG A) be adjustable.

While the frequency of the master DAC clock (`PCG_CLKx_O`) must be synchronous with the sample rate supplied to the external DAC, there is no fixed-phase requirement.

Set the clock divisor and source and low-phase word first, followed by the control register enable bits, which are set together. When the `PCG_PW1/PCG_PW2` registers are set to zero (default), the FS pulse width is (divisor 2) for even divisors and (divisor - 1) 2 for odd divisors. Alternatively, the `PCG_PW1/PCG_PW2` registers can be set high for exactly one-half the period of CLKIN cycles for a 50% duty cycle, provided the FS divisor is an even number.

Clock Configuration Examples

For a `CLKIN` = 33.330 MHz, the two PCGs provide the three synchronous clocks `PCGx_CLK`, serial clock and *FS* for the SRCs and external DAC. These divisors are stored in the `PCG_CTLA1.CLKDIV/PCG_CTLB1.CLKDIV` bit fields.

The integer divisors for several possible sample rates based on 33.330 MHz *CLKIN* are shown in the *Precision Clock Generator Division Ratios* table.

Table 24-2: Precision Clock Generator Division Ratios (33.330 CLKIN)

Sample Rate kHz)	PCG Divisors		FSDIV A* ¹
	CLKDIV B	CLKDIV A	
130.195	1	4	256
<i>65.098</i>	<i>2</i>	<i>8</i>	<i>512</i>
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280
21.699	6	24	1536
18.599	7	28	1792

*1 The frame sync divisor must be an even integer to produce a 50% duty cycle waveform. See [Frame Sync](#).

PCG Event Control

The following sections describe the generation and control of PCG events.

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the `PCG_SYNC1/PCG_SYNC2` registers.

Refer to the *FS Output Synchronization With External Trigger Input* figure. Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

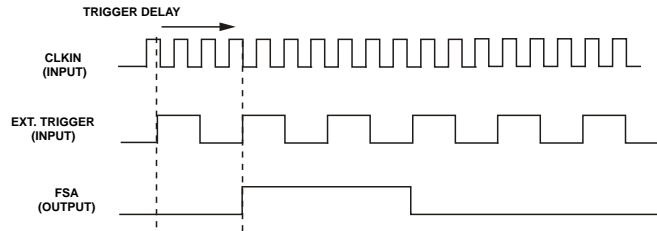


Figure 24-6: FS Output Synchronization With External Trigger Input

External Event Trigger Delay

The time delay between the rising trigger edge and the start of serial clock and frame sync varies between 2.5 to 3.5 input clock periods. If the input clock and the trigger signal are synchronous, the delay is 3 input clock periods. Consider the following cases:

- SCLK0 is the input source. In this case, if the given trigger event is synchronous to SCLK0, the delay is 3 SCLK0 periods. If the trigger signal is asynchronous with SCLK0, the delay varies from 2.5 SCLK0 periods to 3.5 SCLK0 periods. (It depends on whether the trigger edge occurs in the positive half cycle or negative half cycle of SCLK0.)
- The SYS_CLKIN0 is the input source for PCG A, B, C, D. In this case, if the given trigger signal is synchronous to CLKIN, the delay is 3 CLKIN periods. But if they are asynchronous to CLKIN, the delay can vary between 2.5 CLKIN periods to 3.5 CLKIN periods.
- SRU DAI0 is the input source for PCG A, B. SRU DAI1 is the input source for PCG C, D. If the input clock and trigger signal are synchronous, the delay is exactly 3 input clock periods. If asynchronous, it varies between 2.5 to 3.5 input clock periods depending on the phase difference between the input clock and trigger signal.

Programming Model

This section describes the sequence of software steps required for successful PCG operation.

If the PCG is disabled to reprogram a parameter, use a delay after writing to the disable bit. This delay in core clock (*CCLK*) cycles = (PCG source clock period/*CCLK* period). In summary, use the following general procedure:

1. Clear the PCG enable bits without modifying any other settings.
2. Wait for N *CCLK* cycles ($N = \text{PCG source clock period} / \text{processor clock period}$).
3. Program all new parameters without setting the PCG enable bit.
4. Enable the PCG.

Frame Sync Phase Setting

The phase unit requires that the clock and FS are enabled simultaneously in an atomic instruction.

1. Write the clock divider/low 10-bit phase divider to the `PCG_CTLA1/PCG_CTLB1` registers.
2. Program the FS divider/high 10-bit phase divider, enabling both the `PCG_CTLA0.CLKEN/PCG_CTLA0.FSEN` and the `PCG_CTLB0.CLKEN/PCG_CTLB0.FSEN` bits.

Note that both units must be disabled in the same way.

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the `PCG_SYNC` register. Program the phase to 3, so that the rising edge of the external clock is in-sync with the frame sync (*FS Output Synchronization With External Trigger Input*).

Use the following steps.

1. Program the `PCG_SYNC1/PCG_SYNC2` registers and the `PCG_CTLA0` through `PCG_CTLB1` registers appropriately.
2. Enable the clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

Debug Features

Take care in cases where any input to the phase unit is modified. Any individual change of the `PCG_CTLA1.CLKDIV` or `PCG_CTLA0.FSDIV` dividers can cause a failure in the PCG sync operation between the serial clock and the frame sync. Only the programming model ensures a correct setup for phase settings.

ADSP-2156x PCG Register Descriptions

Precision Clock Generator (PCG) contains the following registers.

Table 24-3: ADSP-2156x PCG Register List

Name	Description
<code>PCG_CTLA0</code>	Precision Clock A Control 0 Register
<code>PCG_CTLA1</code>	Precision Clock A Control 1 Register
<code>PCG_CTLB0</code>	Precision Clock B Control 0 Register
<code>PCG_CTLB1</code>	Precision Clock B Control 1 Register
<code>PCG_CTLC0</code>	Precision Clock C Control 0 Register
<code>PCG_CTLC1</code>	Precision Clock C Control 1 Register

Table 24-3: ADSP-2156x PCG Register List (Continued)

Name	Description
PCG_CTLD0	Precision Clock D Control 0 Register
PCG_CTLD1	Precision Clock D Control 1 Register
PCG_PW1	Precision Clock Pulse Width Control 1 Register
PCG_PW2	Precision Clock Pulse Width Control 2 Register
PCG_SYNC1	Precision Clock Frame Sync Synchronization 1 Register
PCG_SYNC2	Precision Clock Frame Sync Synchronization 2 Register

Precision Clock A Control 0 Register

The `PCG_CTLA0` register enables the clock, frame sync, and select divisor for the PCG0 clock A signal.

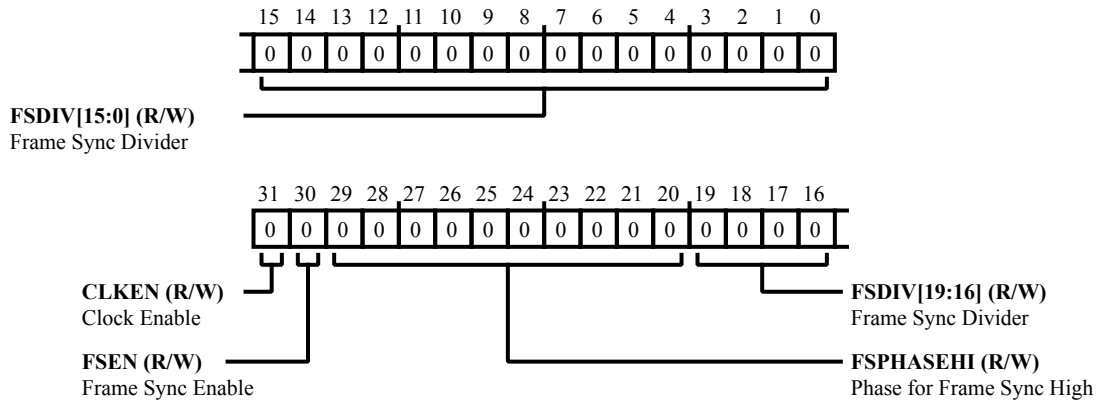


Figure 24-7: PCG_CTLA0 Register Diagram

Table 24-4: PCG_CTLA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKEN	Clock Enable. The <code>PCG_CTLA0 . CLKEN</code> bit enables the clock.
		0 Clock generation disabled
		1 Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The <code>PCG_CTLA0 . FSEN</code> bit enables the frame sync.
		0 Frame sync generation disabled
		1 Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The <code>PCG_CTLA0 . FSPHASEHI</code> bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	FSDIV	Frame Sync Divider. The <code>PCG_CTLA0 . FSDIV</code> bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: <code>PCG_CTLA0 . FSDIV > 1</code> PCGx is in normal mode, <code>PCG_CTLA0 . FSDIV = 0, 1</code> PCGx is in bypass mode.
		0 PCG is in bypass mode
		1 PCG is in bypass mode
		2-1048575 PCG is in normal mode

Precision Clock A Control 1 Register

The `PCG_CTLA1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock A signal.

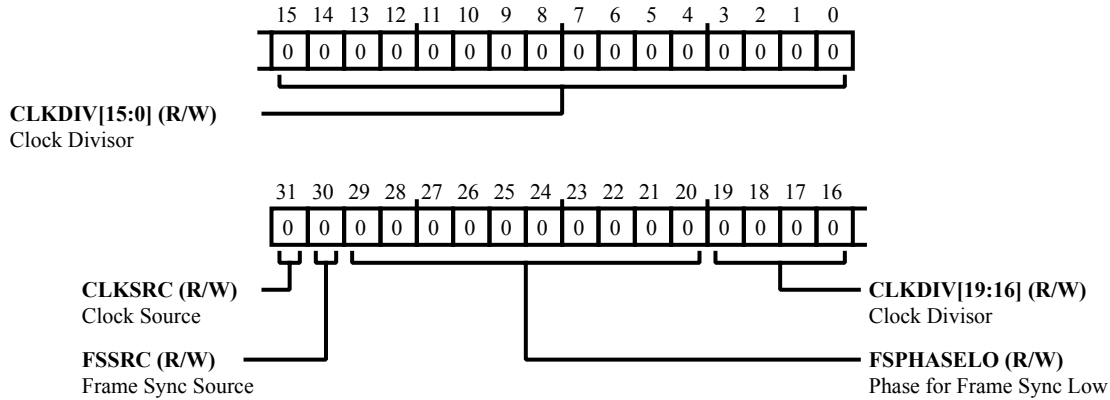


Figure 24-8: `PCG_CTLA1` Register Diagram

Table 24-5: `PCG_CTLA1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKSRC	Clock Source. The <code>PCG_CTLA1 . CLKSRC</code> bit specifies the clock source.
		0 CLKIN0 pin selected for clock
		1 <code>PCG_EXT_DAI0</code> selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The <code>PCG_CTLA1 . FSSRC</code> bit specifies the frame sync source.
		0 CLKIN0 pin selected for frame sync
		1 <code>PCG_EXTX_DAI0</code> selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The <code>PCG_CTLA1 . FSPHASELO</code> bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	CLKDIV	Clock Divisor. The <code>PCG_CTLA1 . CLKDIV</code> bit field contains the clock divisor value.

Precision Clock B Control 0 Register

The `PCG_CTLB0` register enables the clock, frame sync, and select divisor for the PCG0 clock B signal.

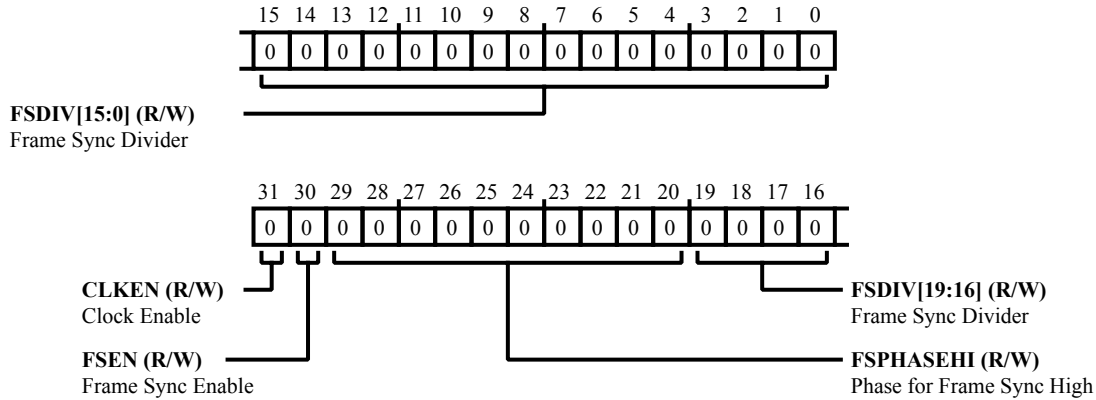


Figure 24-9: PCG_CTLB0 Register Diagram

Table 24-6: PCG_CTLB0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKEN	Clock Enable. The <code>PCG_CTLB0.CLKEN</code> bit enables the clock.
		0 Clock generation disabled
		1 Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The <code>PCG_CTLB0.FSEN</code> bit enables the frame sync.
		0 Frame sync generation disabled
		1 Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The <code>PCG_CTLB0.FSPHASEHI</code> bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	FSDIV	Frame Sync Divider. The <code>PCG_CTLB0.FSDIV</code> bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: <code>PCG_CTLB0.FSDIV > 1</code> PCGx is in normal mode, <code>PCG_CTLB0.FSDIV = 0, 1</code> PCGx is in bypass mode.
		0 PCG is in bypass mode
		1 PCG is in bypass mode
		2-1048575 FSDIV > 1 PCG is in normal mode

Precision Clock B Control 1 Register

The `PCG_CTLB1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock B signal.

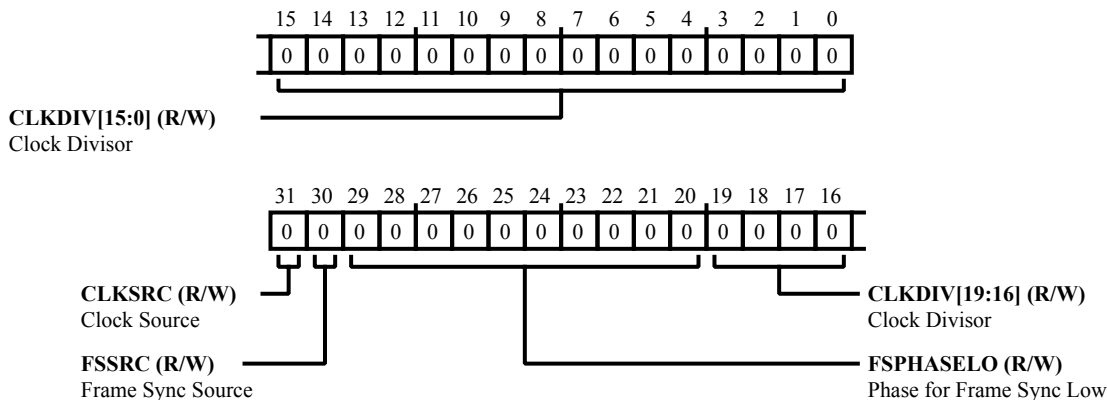


Figure 24-10: PCG_CTLB1 Register Diagram

Table 24-7: PCG_CTLB1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKSRC	Clock Source. The <code>PCG_CTLB1 . CLKSRC</code> bit specifies the clock source.
		0 CLKIN0 pin selected for clock
		1 PCG_EXT_DAI0 selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The <code>PCG_CTLB1 . FSSRC</code> bit specifies the frame sync source.
		0 CLKIN0 pin selected for frame sync
		1 PCG_EXT_DAI0 selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The <code>PCG_CTLB1 . FSPHASELO</code> bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	CLKDIV	Clock Divisor. The <code>PCG_CTLB1 . CLKDIV</code> bit field contains the clock divisor value.

Precision Clock C Control 0 Register

The `PCG_CTLC0` register enables the clock, frame sync, and select divisor for the PCG0 clock C signal.

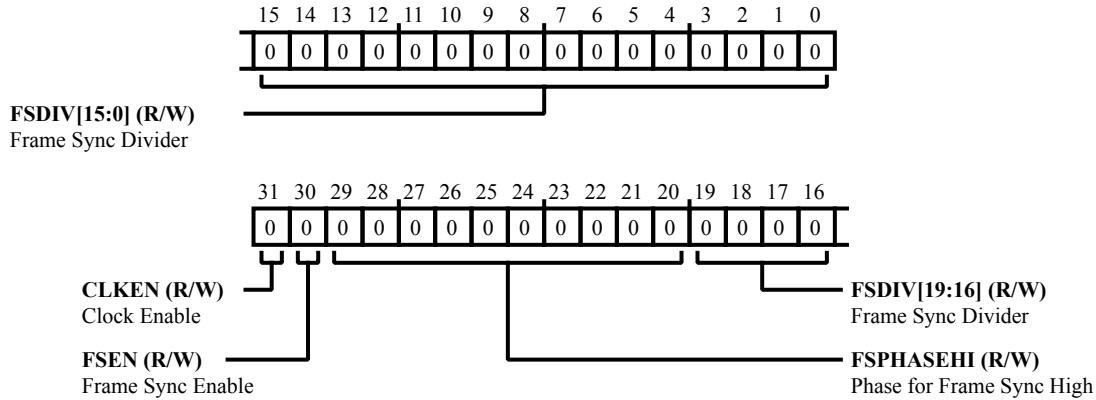


Figure 24-11: PCG_CTLC0 Register Diagram

Table 24-8: PCG_CTLC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKEN	Clock Enable. The <code>PCG_CTLC0 . CLKEN</code> bit enables the clock.
		0 Clock generation disabled
		1 Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The <code>PCG_CTLC0 . FSEN</code> bit enables the frame sync.
		0 Frame sync generation disabled
		1 Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The <code>PCG_CTLC0 . FSPHASEHI</code> bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	FSDIV	Frame Sync Divider. The <code>PCG_CTLC0 . FSDIV</code> bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: <code>PCG_CTLC0 . FSDIV > 1</code> PCGx is in normal mode, <code>PCG_CTLC0 . FSDIV = 0, 1</code> PCGx is in bypass mode.
		0 PCG is in bypass mode
		1 PCG is in bypass mode
		2-1048575 FSDIV > 1 PCG is in normal mode

Precision Clock C Control 1 Register

The `PCG_CTLC1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock C signal.

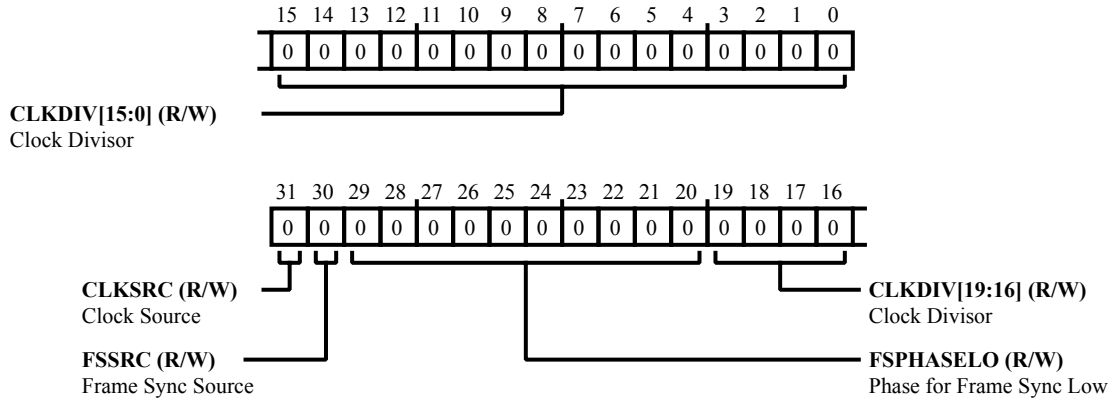


Figure 24-12: `PCG_CTLC1` Register Diagram

Table 24-9: `PCG_CTLC1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKSRC	Clock Source. The <code>PCG_CTLC1 . CLKSRC</code> bit specifies the clock source.
		0 CLKIN1 pin selected for clock
		1 <code>PCG_EXT_DAI1</code> selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The <code>PCG_CTLC1 . FSSRC</code> bit specifies the frame sync source.
		0 CLKIN1 pin selected for frame sync
		1 <code>PCG_EXT_DAI1</code> selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The <code>PCG_CTLC1 . FSPHASELO</code> bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	CLKDIV	Clock Divisor. The <code>PCG_CTLC1 . CLKDIV</code> bit field contains the clock divisor value.

Precision Clock D Control 0 Register

The `PCG_CTLD0` register enables the clock, frame sync, and select divisor for the PCG0 clock D signal.

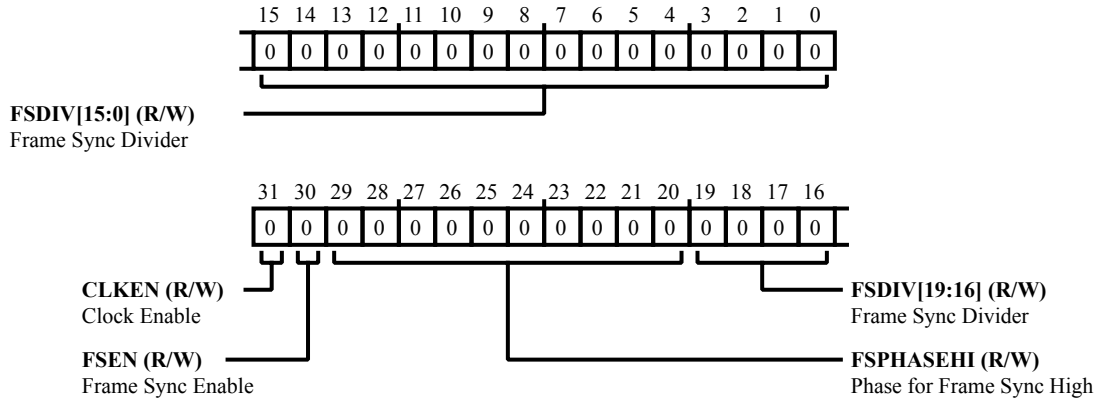


Figure 24-13: PCG_CTLD0 Register Diagram

Table 24-10: PCG_CTLD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKEN	Clock Enable. The <code>PCG_CTLD0.CLKEN</code> bit enables the clock.
		0 Clock generation disabled
		1 Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The <code>PCG_CTLD0.FSEN</code> bit enables the frame sync.
		0 Frame sync generation disabled
		1 Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The <code>PCG_CTLD0.FSPHASEHI</code> bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	FSDIV	Frame Sync Divider. The <code>PCG_CTLD0.FSDIV</code> bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: <code>PCG_CTLD0.FSDIV > 1</code> PCGx is in normal mode, <code>PCG_CTLD0.FSDIV = 0, 1</code> PCGx is in bypass mode.
		0 PCG is in bypass mode
		1 PCG is in bypass mode
		2-1048575 FSDIV > 1 PCG is in normal mode

Precision Clock D Control 1 Register

The `PCG_CTLD1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock D signal.

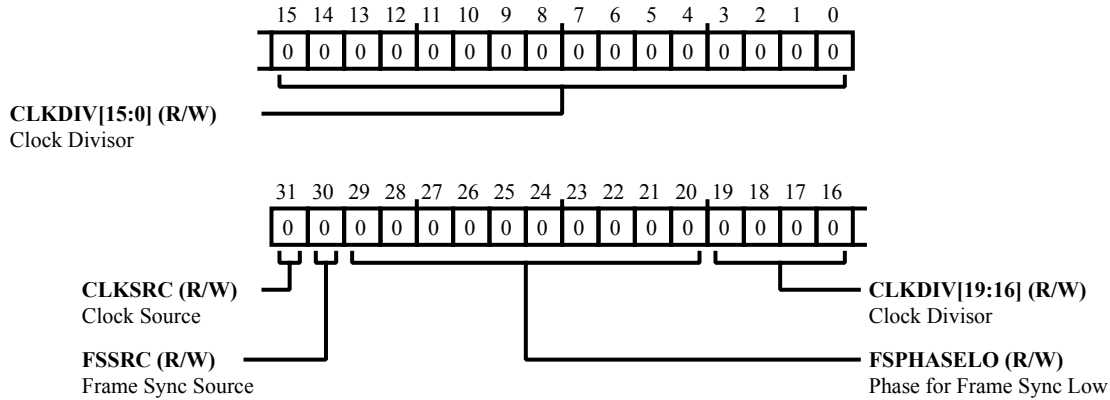


Figure 24-14: PCG_CTLD1 Register Diagram

Table 24-11: PCG_CTLD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CLKSRC	Clock Source. The <code>PCG_CTLD1 . CLKSRC</code> bit specifies the clock source.
		0 CLKIN1 pin selected for clock
		1 PCG_EXT_DAI1 selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The <code>PCG_CTLD1 . FSSRC</code> bit specifies the frame sync source.
		0 CLKIN1 pin selected for frame sync
		1 PCG_EXT_DAI1 selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The <code>PCG_CTLD1 . FSPHASELO</code> bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.
19:0 (R/W)	CLKDIV	Clock Divisor. The <code>PCG_CTLD1 . CLKDIV</code> bit field contains the clock divisor value.

Precision Clock Pulse Width Control 1 Register

The `PCG_PW1` register sets the one shot frame sync and the active low frame sync select for PCG A and PCG B.

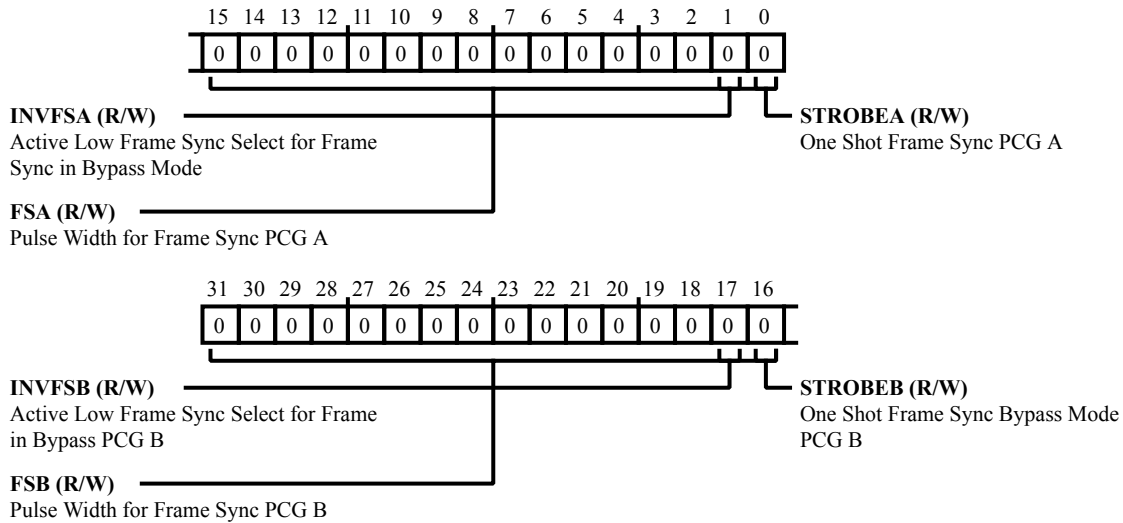


Figure 24-15: PCG_PW1 Register Diagram

Table 24-12: PCG_PW1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	INVFSB	Active Low Frame Sync Select for Frame in Bypass PCG B. The <code>PCG_PW1</code> . <code>INVFSB</code> bit selects active low or active high frame sync in bypass mode for PCG B.
16 (R/W)	STROBEB	One Shot Frame Sync Bypass Mode PCG B. The <code>PCG_PW1</code> . <code>STROBEB</code> bit sets the frame sync pulse in bypass mode for PCG B. This is the duration equal to one period of the <code>DAI_MISCA2_I</code> signal (PCG B) repeating at the beginning of every frame.
31:16 (R/W)	FSB	Pulse Width for Frame Sync PCG B. The <code>PCG_PW1</code> . <code>FSB</code> bit field sets the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync.
1 (R/W)	INVFSA	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The <code>PCG_PW1</code> . <code>INVFSA</code> bit selects active low or active high frame sync for PCG A in bypass mode.
15:0 (R/W)	FSA	Pulse Width for Frame Sync PCG A. The <code>PCG_PW1</code> . <code>FSA</code> bit field sets the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync.

Table 24-12: PCG_PW1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	STROBEA	One Shot Frame Sync PCG A. The PCG_PW1 . STROBEA bit sets the frame sync pulse for PCG A in bypass mode. This is the duration equal to one period of the DAI_MISCA2_I signal (PCG A) repeating at the beginning of every frame.

Precision Clock Pulse Width Control 2 Register

The `PCG_PW2` register sets the one shot frame sync and the active low frame sync select for PCG C and PCG D.

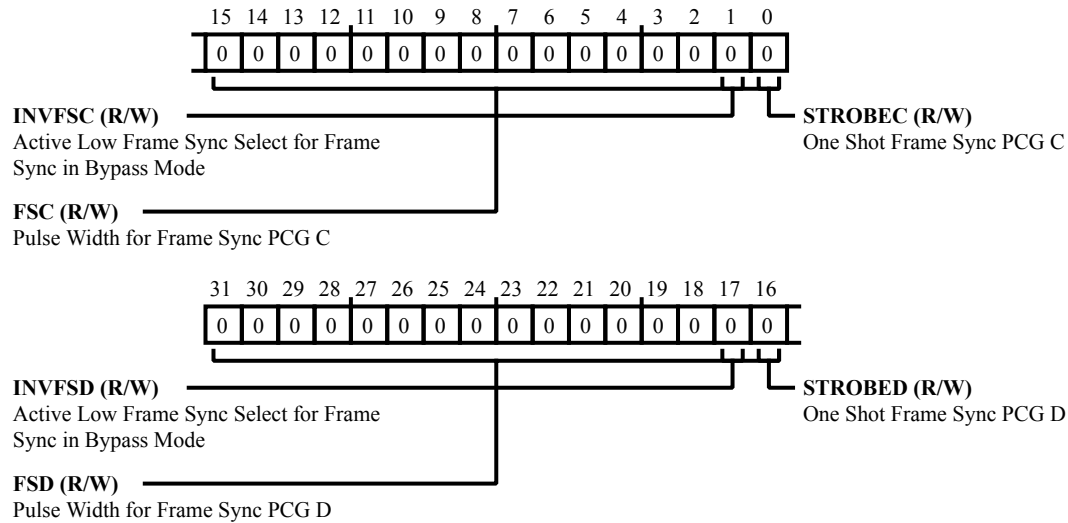


Figure 24-16: PCG_PW2 Register Diagram

Table 24-13: PCG_PW2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	INVFS D	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The <code>PCG_PW2</code> . <code>INVFS D</code> bit selects active low or active high frame sync for PCG D in bypass mode.
16 (R/W)	STROBED	One Shot Frame Sync PCG D. The <code>PCG_PW2</code> . <code>STROBED</code> bit sets the frame sync pulse for PCG D in bypass mode. This is the duration equal to one period of the <code>DAI_MISCA4_I</code> signal (PCG D) repeating at the beginning of every frame.
31:16 (R/W)	FSD	Pulse Width for Frame Sync PCG D. The <code>PCG_PW2</code> . <code>FSD</code> bit field sets the number of input clock periods for which the frame sync output is high for PCG D. Pulse width should be less than the divisor of the frame sync.
1 (R/W)	INVFSC	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The <code>PCG_PW2</code> . <code>INVFSC</code> bit selects active low or active high frame sync for PCG C in bypass mode.
15:0 (R/W)	FSC	Pulse Width for Frame Sync PCG C. The <code>PCG_PW2</code> . <code>FSC</code> bit field sets the number of input clock periods for which the frame sync output is high for PCG C. Pulse width should be less than the divisor of the frame sync.

Table 24-13: PCG_PW2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	STROBEC	One Shot Frame Sync PCG C. The PCG_PW2 . STROBEC bit sets the frame sync pulse for PCG C in bypass mode. This is the duration equal to one period of the DAI_MISCA3_I signal (PCG C) repeating at the beginning of every frame.

Precision Clock Frame Sync Synchronization 1 Register

The `PCG_SYNC1` register allows programs to synchronize the clock frame sync units with external frame syncs. Note that the `PCG_CTLA1.CLKSRC` bit is overridden if `PCG_SYNC1.CLKASRC` bit in the `PCG_SYNC1` register is set.

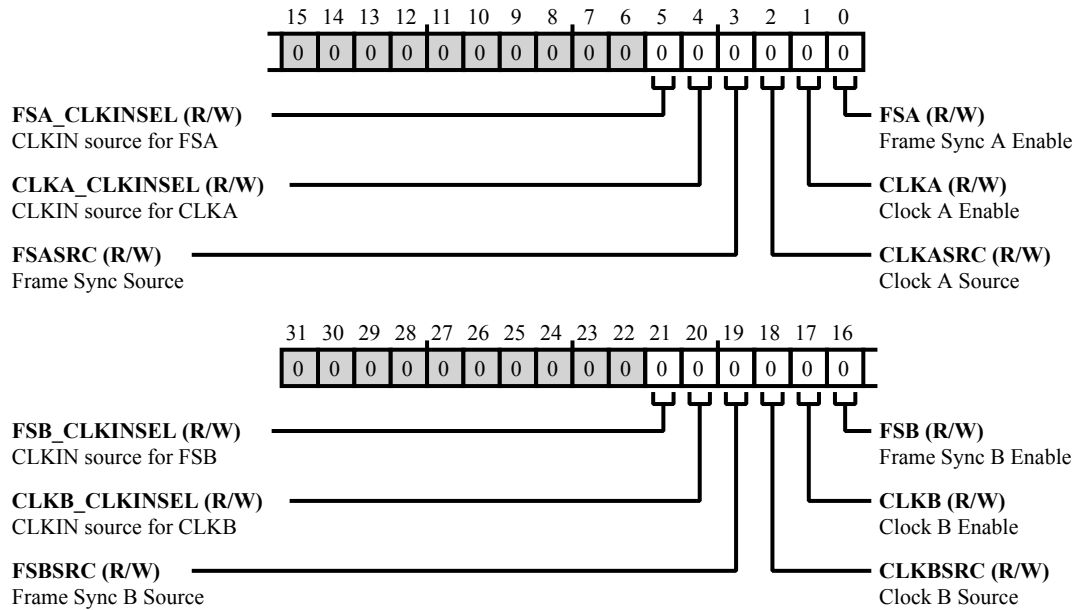


Figure 24-17: `PCG_SYNC1` Register Diagram

Table 24-14: `PCG_SYNC1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	FSB_CLKINSEL	CLKIN source for FSB. The <code>PCG_SYNC1.FSB_CLKINSEL</code> bit enables the CLKIN input source for FSB
		0 Selects CLKIN0 source for FSB CLKIN
		1 Selects CLKIN0 source for FSB CLKIN
20 (R/W)	CLKB_CLKINSEL	CLKIN source for CLKB. The <code>PCG_SYNC1.CLKB_CLKINSEL</code> bit enables the CLKIN input source for CLKB
		0 Select CLKIN0 source for CLKB CLKIN
		1 Select CLKIN0 source for CLKB CLKIN

Table 24-14: PCG_SYNC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	FSBSRC	Frame Sync B Source. The PCG_SYNC1.FSBSRC bit enables the frame sync B input source.
		0 Output selected by FSBSOURCE bit
		1 Clock derived from core PLL selected for frame sync B
18 (R/W)	CLKBSRC	Clock B Source. The PCG_SYNC1.CLKBSRC bit enables the clock B input source.
		0 Output selected by CLKBSOURCE bit
		1 Clock derived from core PLL selected for clock B
17 (R/W)	CLKB	Clock B Enable. The PCG_SYNC1.CLKB bit enables synchronization of clock B with the external frame sync.
		0 Clock disabled
		1 Clock enabled
16 (R/W)	FSB	Frame Sync B Enable. The PCG_SYNC1.FSB bit enables synchronization of frame sync B with the external frame sync.
		0 Frame sync disabled
		1 Frame sync enabled
5 (R/W)	FSA_CLKINSEL	CLKIN source for FSA. The PCG_SYNC1.FSA_CLKINSEL bit enables the CLKIN input source for FSA
		0 Selects CLKIN0 as source for FSA CLKIN
		1 Selects CLKIN0 as source for FSA CLKIN
4 (R/W)	CLKA_CLKINSEL	CLKIN source for CLKA. The PCG_SYNC1.CLKA_CLKINSEL bit enables the CLKIN input source for CLKA
		0 Select CLKIN0 source for CLKA CLKIN
		1 Select CLKIN0 source for CLKA CLKIN
3 (R/W)	FSASRC	Frame Sync Source. The PCG_SYNC1.FSASRC bit enables the frame sync A input source.
		0 Output selected by FSASOURCE bit
		1 Clock derived from core PLL selected for frame sync A

Table 24-14: PCG_SYNC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	CLKASRC	Clock A Source. The PCG_SYNC1 . CLKASRC bit enables the clock A input source.
		0 Output selected by CLKASOURCE bit
		1 Clock derived from core PLL selected for clock A
1 (R/W)	CLKA	Clock A Enable. The PCG_SYNC1 . CLKA bit enables synchronization of clock A with the external frame sync.
		0 Clock disabled
		1 Clock enabled
0 (R/W)	FSA	Frame Sync A Enable. The PCG_SYNC1 . FSA bit enables synchronization of frame sync A with the external frame sync.
		0 Frame Sync Disabled
		1 Frame Sync Enabled

Precision Clock Frame Sync Synchronization 2 Register

The `PCG_SYNC2` register allows programs to synchronize the clock frame sync units with external frame syncs. Note that the `PCG_CTLD1.CLKSRC` bit is overridden if `PCG_SYNC2.CLKDSRC` bit in the `PCG_SYNC2` register is set.

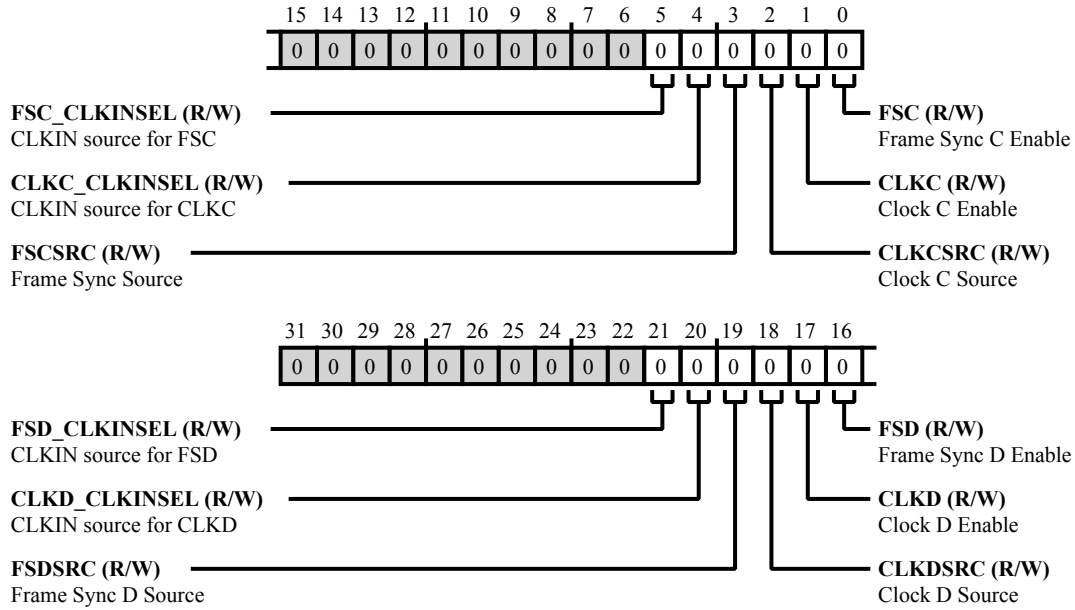


Figure 24-18: `PCG_SYNC2` Register Diagram

Table 24-15: `PCG_SYNC2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	<code>FSD_CLKINSEL</code>	CLKIN source for FSD. The <code>PCG_SYNC2.FSD_CLKINSEL</code> bit enables the CLKIN input source for FSD
		0 Selects CLKIN0 source for FSB CLKIN
		1 Selects CLKIN0 source for FSB CLKIN
20 (R/W)	<code>CLKD_CLKINSEL</code>	CLKIN source for CLKD. The <code>PCG_SYNC2.CLKD_CLKINSEL</code> bit enables the CLKIN input source for CLKD
		0 Select CLKIN0 source for CLKB CLKIN
		1 Select CLKIN0 source for CLKB CLKIN

Table 24-15: PCG_SYNC2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	FSDSRC	Frame Sync D Source. The PCG_SYNC2 . FSDSRC bit enables the frame sync D input source.
		0 Output selected by FSDSOURCE bit
		1 Clock derived from core PLL selected for frame sync D
18 (R/W)	CLKDSRC	Clock D Source. The PCG_SYNC2 . CLKDSRC bit enables the clock D input source.
		0 Output selected by CLKDSOURCE bit
		1 Clock derived from core PLL selected for clock D
17 (R/W)	CLKD	Clock D Enable. The PCG_SYNC2 . CLKD bit enables synchronization of clock D with the external frame sync.
		0 Clock disabled
		1 Clock enabled
16 (R/W)	FSD	Frame Sync D Enable. The PCG_SYNC2 . FSD bit enables synchronization of frame sync D with the external frame sync.
		0 Frame sync disabled
		1 Frame sync enabled
5 (R/W)	FSC_CLKINSEL	CLKIN source for FSC. The PCG_SYNC2 . FSC_CLKINSEL bit enables the CLKIN input source for FSC
		0 Selects CLKIN0 as source for FSA CLKIN
		1 Selects CLKIN0 as source for FSA CLKIN
4 (R/W)	CLKC_CLKINSEL	CLKIN source for CLKC. The PCG_SYNC2 . CLKC_CLKINSEL bit enables the CLKIN input source for CLKC
		0 Select CLKIN0 source for CLKA CLKIN
		1 Select CLKIN0 source for CLKA CLKIN
3 (R/W)	FSCSRC	Frame Sync Source. The PCG_SYNC2 . FSCSRC bit enables the frame sync C input source.
		0 Output selected by FSCSOURCE bit
		1 Clock derived from core PLL selected for frame sync C

Table 24-15: PCG_SYNC2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	CLKCSRC	Clock C Source. The PCG_SYNC2.CLKCSRC bit enables the clock C input source.
		0 Output selected by CLKCSOURCE bit
		1 Clock derived from core PLL selected for clock C
1 (R/W)	CLKC	Clock C Enable. The PCG_SYNC2.CLKC bit enables synchronization of clock C with the external frame sync.
		0 Clock disabled
		1 Clock enabled
0 (R/W)	FSC	Frame Sync C Enable. The PCG_SYNC2.FSC bit enables synchronization of frame sync C with the external frame sync.
		0 Frame sync disabled
		1 Frame sync enabled

25 Asynchronous Sample Rate Converter (ASRC)

Sample rate converters (SRC) are frequently used in digital signal processing audio applications. The most frequently used sample rate conversions are off-loaded into hardware modules that are dedicated for filter processing and reduce the instruction processing load on the core, freeing it up for other tasks.

Features

The ASRC has these features and capabilities.

- 4 asynchronous stereo SRCs operating in slave mode are available in each DAI
- Simple programming model
- Controllable muting options (hardware, software and automatic)
- Automatically senses input and output sample frequencies
- Supports left-justified, I²S, right-justified (16-,18-, 20-, 24-bits), and TDM serial port modes
- Daisy-chain configuration in TDM modes (including between DAI0 and DAI1) for input and output ports to create a serial frame
- Different protocols on input/output port allow format conversions
- De-emphasis filter for 32, 44.1 and 48 kHz sampling frequencies
- Up to 192 kHz sample rate input/output continuous sample ratios from 7.5:1 to 1:8
- Group delay (latency of interpolation filter) is 16 samples
- SNR from 128 to 140 dB (depending on processor model)
- Matched phase mode available to compensate for group delays
- Can be used to de-jitter clocks in systems

Functional Description

Conceptually, the sample rate converter interpolates the serial input data at a rate of 220 and samples the interpolated data stream by the output sample rate. In practice, a 64-tap FIR filter with 220 polyphases, a FIFO, a digital servo loop that measures the time difference between the input and output samples within 5 ps, and a digital circuit to track the sample rate ratio are used to perform the interpolation and output sampling.

ADSP-2156x ASRC Register List

Sample Rate Converter Module

Table 25-1: ADSP-2156x ASRC Register List

Name	Description
ASRC_CTL01	Control Register for ASRC 0 and 1
ASRC_CTL23	Control Register for ASRC 2 and 3
ASRC_MUTE	Mute Register
ASRC_RAT01	Ratio Register for ASRC 0 and 1
ASRC_RAT23	Ratio Register for ASRC 2 and 3

ASRC Interrupt List

The ASRC interrupts are controlled through the DAI.

Table 25-2: ASRC Interrupt List

Interrupt ID	Interrupt Name	Interrupt Condition
20	DAI0_IRQH	ASRC initialization
21	DAI1_IRQH	ASRC sample rate change
116	DAI0_IRQL	
117	DAI1_IRQL	

ASRC Block Diagram

The *ASRC Block Diagram* figure shows a top level block diagram of the ASRC module and the *Core Architecture* figure shows architecture details.

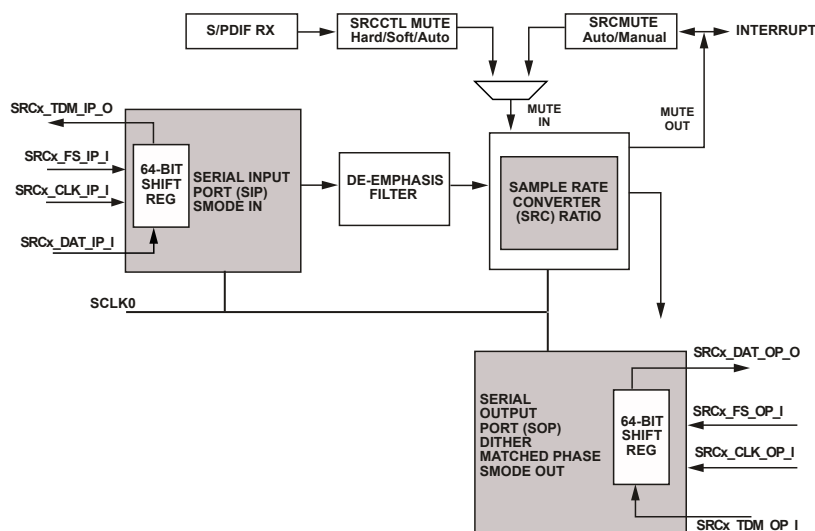


Figure 25-1: ASRC Block Diagram

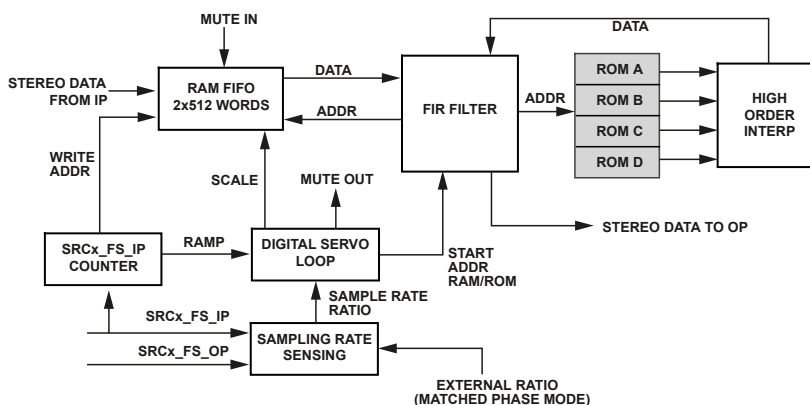


Figure 25-2: ASRC Core Architecture

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the ASRCs to the output pins or any other peripherals. For more information, see the [Digital Audio Interface \(DAI\)](#) chapter.

Clocking

The ASRC module is in the SCLK0 clock domain. An internal divided version of the SCLK0 clock is generated and used as the fundamental clock for the ASRC module.

I/O Ports

The I/O ports provide the interface through which data is transferred asynchronously into and out of the SRC modules. The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I²S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support TDM mode for daisy-chaining

multiple SRCs to form a frame. The serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected.

NOTE: The SRC converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

De-Emphasis Filter

The de-emphasis filter is used to de-emphasize audio data that has been emphasized.

Mute Control

When either the SRC starts up (or there is a change in sample ratio), the mute out signal (`SRCx_MUTEOUT`) is asserted (=1). The mute out signal stays high until the SRC settles on the new sample rates. While mute out is asserted high, the mute in signal should be asserted high as well. The mute in signal performs a soft mute of the audio input data when asserted and un-mutes the input audio data softly when deasserted.

Note that it takes 4096 input port FS samples until the audio input data is completely muted and 4096 FS samples until the audio input data is completely unmuted.

SRC Core

As shown in the *ASRC Core Architecture* figure, the sample rate converter's RAM FIFO block adjusts the left and right input samples and stores them for the FIR filter's convolution cycle. The `ASRCx_FS_IP` counter provides the write address (for scaling) to the FIFO block and the ramp input to the digital-servo loop. The ROM stores the coefficients for the FIR filter convolution and performs a high-order interpolation between the stored coefficients. The sample rate ratio block measures the sample rate by dynamically altering the ROM coefficients and scaling the FIR filter length and input data. The digital-servo loop automatically tracks the `SRCx_FS_IP` and `SRCx_FS_OP` sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.

NOTE: Unlike other peripherals, the sample rate converters own local memories (RAM and ROM) which are dedicated for the purpose of sample rate conversion only.

The sample rate converter only operates asynchronously and is always a slave to the input and output ports.

RAM FIFO

The RAM FIFO receives the left and right input data and adjusts the amplitude of the data for both the soft muting of the SRC and the scaling of the input data by the sample rate ratio before storing the samples in RAM. The input data is scaled by the sample rate ratio because as the FIR filter length of the convolution increases, so does the amplitude of the convolution output. To keep the output of the FIR filter from saturating, the input data is scaled down by multiplying it by $(\text{SRCx_FS_OP})/(\text{SRCx_FS_IP})$ when $\text{SRCx_FS_OP} < \text{SRCx_FS_IP}$. The FIFO also scales the input data to mute and stop muting the SRC.

Digital Servo Loop

The digital-servo loop is essentially a ramp filter that provides the initial pointer to the address in RAM and ROM for the start of the FIR convolution. The RAM pointer is the integer output of the ramp filter while the ROM pointer is the fractional part. The digital-servo loop must be able to provide excellent rejection of jitter on the SRCx_FS_IP and SRCx_FS_OP clocks as well as measure the arrival of the SRCx_FS_OP clock within 5 ps. The digital-servo loop also divides the fractional part of the ramp output by the ratio of (SRCx_FS_IP)/(SRCx_FS_OP) for the case when SRCx_FS_IP > SRCx_FS_OP, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample rate, a fast mode has been added to the filter. When the digital-servo loop starts up or the sample rate is changed, the digital-servo loop kicks into fast mode to adjust and settle on the new sample rate. Upon sensing the digital-servo loop settling down to some reasonable value, the digital-servo loop kicks into normal or slow mode. During fast mode, the SRCx_MUTE_OUT bit of the ASRC is asserted to mute the ASRC input which avoids clicks and pops.

FIR Filter

The FIR filter is a 64-tap filter in the case of SRCx_FS_OP < SRCx_FS_IP and is (SRCx_FS_IP)/(SRCx_FS_OP) × 64 taps for the case when SRCx_FS_IP > SRCx_FS_OP. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the SRCx_FS_OP period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the (SRCx_FS_OP/SRCx_FS_IP) × 2²⁰ ratio for SRCx_FS_IP > SRCx_FS_OP or 2²⁰ for SRCx_FS_OP < SRCx_FS_IP. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

Sample Rate Sensing

The (SRCx_FS_IP)/(SRCx_FS_OP) sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when SRCx_FS_IP > SRCx_FS_OP. The ratio is calculated by comparing the output of an SRCx_FS_OP counter to the output of an SRCx_FS_IP counter. If SRCx_FS_OP > SRCx_FS_IP, the ratio is held at one. If SRCx_FS_IP > SRCx_FS_OP, the sample rate ratio is updated if it is different by more than two SRCx_FS_OP periods from the previous SRCx_FS_OP to SRCx_FS_IP comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

Digital Filter Group Delay

The RAM in the FIFO is 512 words deep for both left and right channels. An offset of 16 samples to the write address, provided by the SRCx_FS_IP counter, is added to prevent the RAM read pointer from overlapping the write address. The maximum decimation rate can be calculated from the RAM word: depth = (512 – 16) ÷ 64 taps = 7.75:1.

The 64 samples effect latency in the interpolation filter. This latency (group delay) depends on interpolation or decimation ratio and is determined as follows:

Interpolation or Decimation Ratio (1): $GDL = 16/f_{S_IN} + 32/f_{S_IN}$ seconds for SRC_FS_OP > SRC_FS_IP

Interpolation or Decimation Ratio (2): $GDL = 16/f_{S_IN} + 32/f_{S_IN} \times f_{S_IN}/f_{S_OUT}$ seconds for $SRC_FS_OP < SRC_FS_IP$

Data Format

The *ASRC Data Frame Format by Protocol* figure shows the data input format for a frame (stereo data). The frame format is valid for all protocols. For models which do not support matched phase mode the 8-bit data field is ignored.

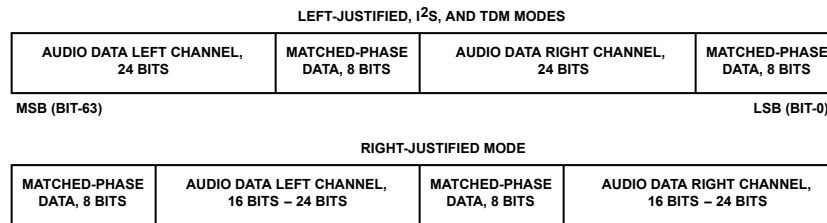


Figure 25-3: ASRC Data Frame Format by Protocol

Operating Modes

The ASRC can operate in TDM, I²S, left-justified, right-justified, and bypass modes. The serial ports of the processor can be used for moving the ASRC data to/from the internal memory.

In I²S, left-justified and right-justified modes, the ASRCs operate individually. The serial data provided in the input port is converted to the sample rate of the output port.

TDM Input Mode

In TDM input port, several ASRCs can be daisy-chained together and connected to the serial input port of a SHARC processor or other processor (see the *TDM Input/Output Modes* figure). The ASRC IP contains a 64-bit parallel load shift register. When the $SRCx_FS_IP_I$ pulse arrives, each ASRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to $SRCx_DATA_IP_I$, while the output is connected to $SRCx_TDM_IP_O$. By connecting the $SRCx_TDM_IP_O$ to the $SRCx_DATA_IP_I$ of the next ASRC, a large shift register is created, which is clocked by $SRCx_CLK_IP_I$.

NOTE: In TDM mode, the ASRC drives at the rising edge and samples at the falling edge of the serial clock. In all other modes, the serial clock rising edge is the sampling edge, and the falling edge is the driving edge.

NOTE: The number of ASRCs that can be daisy-chained together is limited by the maximum frequency of $SRCx_CLK_xx_I$, refer to the data sheet for exact values. For example, if the maximum frequency of $SRCx_CLK_xx_I$ is x MHz, and the output sample rate is f_S , then number of ASRCs (n) that can be connected in daisy chained fashion is: $n \cdot 64 \cdot f_S \leq x$ MHz.

TDM Output Mode

As shown in the *TDM Input/Output Modes* figure, using the TDM output port several ASRCs can be daisy-chained together and connected to the SPORT of this or another processor. The ASRC OP contains a 64-bit parallel load shift register. When the SRC_x_FS_OP_I pulse arrives, each ASRC loads its left and right data into the 64-bit shift register. The input to the shift register is connected to SRC_x_TDM_OP_I, and the output is connected to SRC_x_DAT_OP_O. By connecting the SRC_x_DAT_OP_O to the SRC_x_TDM_OP_I of the next ASRC, a large shift register is created, which is clocked by SRC_x_CLK_OP_I.

As shown in *TDM Input/Output Modes*, with three ASRCs in a daisy-chain connection, the serial clock for input/output port is defined as: SCLK = 3 × 64 FS = 192 FS.

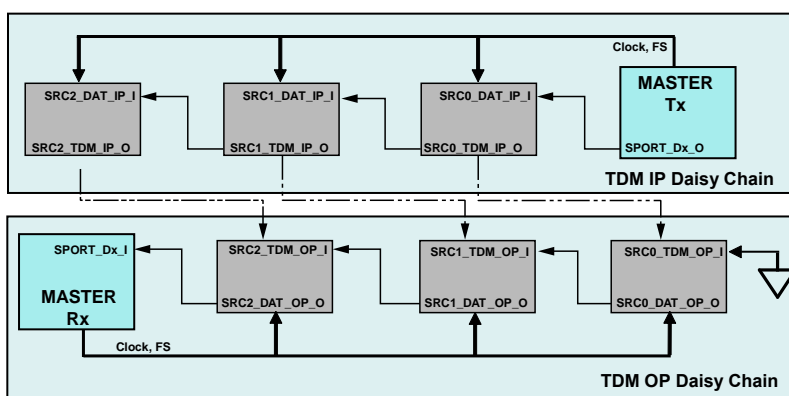


Figure 25-4: TDM Input/Output Modes

Matched-Phase Mode

The matched-phase mode of the sample rate converter, shown in *Typical Configuration for Matched-Phase Mode Operation*, is enabled by the ASRC_CTL01.MPHASE0, ASRC_CTL01.MPHASE1, ASRC_CTL23.MPHASE2 and ASRC_CTL23.MPHASE3 bits. This mode is used to match the phase (group delay) between two or more adjacent sample rate converters that are operating with the same input and output clocks.

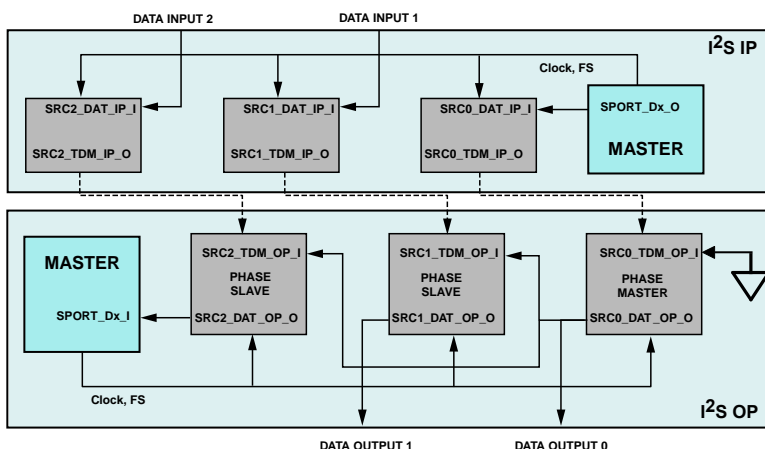


Figure 25-5: Typical Configuration for Matched-Phase Mode Operation

Hysteresis of the $(SRCx_FS_OP)/(SRCx_FS_IP)$ ratio circuit can cause phase mismatching between two ASRCs operating with the same input and output clocks. Since the hysteresis requires a difference of more than two $SRCx_FS_OP$ periods to update the $SRCx_FS_OP$ and $SRCx_FS_IP$ ratios, two ASRCs may have differences in their ratios from 0 to 4 $SRCx_FS_OP$ period counts. The $(SRCx_FS_OP)/(SRCx_FS_IP)$ ratio adjusts the filter length of the ASRC, which corresponds directly with the group delay. Thus, the magnitude in the phase difference depends upon the resolution of the $SRCx_FS_OP$ and $SRCx_FS_IP$ counters. The greater the resolution of the counters, the smaller the phase difference error.

When the slave SRC matched-phase mode bit is set (=1), it accepts the sample rate ratio transmitted by another SRC, (the matched-phase master) which has its matched-phase mode bit cleared (=0), through its serial output.

The phase master ASRC device transmits its $SRCx_FS_OP/SRCx_FS_IP$ ratio through the data output pin ($SRCx_DAT_OP_O$) to the slave's ASRC's data input pins ($SRCx_TDM_OP_I$). The transmitted data (32-bit subframe) contains 24-bit data and 8-bits matched phase (see the *ASRC Data Frame Format by Protocol* figure).

The slave SRCs receive the 8-bit matched phase bits (instead of their own internally-derived ratio) if their $SRCx_MPHASE$ bits are set to 1, respectively. The $SRCx_FS_IP$ and $SRCx_FS_OP$ signals may be asynchronous with respect to each other in this mode. Note that there must be 64 $SRCx_CLK_OP$ cycles per frame in matched-phase mode (two 24-bits data and two 8-bits phase match).

NOTE: By default, matched phased data is sent on the $SRCx_DAT_OP_O$ pin, but only if the $SRCx_TDM_OP_I$ pin is tied low. The slaves simply ignore the matched phased data if their $ASRC_CTL01.MPHASE1$ through $ASRC_CTL23.MPHASE3$ bits are cleared (= 0).

Bypass Mode

When the $ASRC_CTL01.BYP0$, $ASRC_CTL01.BYP1$, $ASRC_CTL23.BYP2$ and $ASRC_CTL23.BYP3$ bits are set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. Dithering is disabled. This mode is ideal when the input and output sample rates are the same and the $SRCx_FS_IP_I$ and $SRCx_FS_OP_I$ signals are synchronous with respect to each other. In matched phase bypass mode, the $SRCx_FS_OP_I$ signal should come at least one $SRCx_CLK_xx_I$ period before $SRCx_FS_IP_I$. Cases where this is not met could result in data loss. For example, if internal SPORTS are used then the $SRCx_FS_OP_I$ and $SRCx_FS_IP_I$ signals could be driven by different SPORTs so that the timing of these signals can be controlled by enabling them at different times. This mode can also be used for passing through non-audio data since no processing is performed on the input data.

De-Emphasis Mode

The $ASRC_CTL01.DEEMPHASIS0$, $ASRC_CTL01.DEEMPHASIS1$, $ASRC_CTL23.DEEMPHASIS2$ and $ASRC_CTL23.DEEMPHASIS3$ bits choose the type of de-emphasis filter based on the input sample rate for 32, 44.1 or 48 kHz sampling rates.

Dithering Mode

The $ASRC_CTL01.DITHER0$, $ASRC_CTL01.DITHER1$, $ASRC_CTL23.DITHER2$, and $ASRC_CTL23.DITHER3$ control this mode of operation. Serial output data is dithered down to 20, 18, or 16 bits

when 20-, 18-, or 16-bit output data is selected. In the case of 20-, 18- and 16-bit word lengths, the least significant bits of the 24-bit word coming from the SRC into the serial output port are truncated. The `DITHER_EN` signal (not user configurable) automatically adds dithering to the 24-bit word before truncating to the appropriate output word length. The `21BIT_DITHER` signal is used for the consumer version of the SRC to reduce the dynamic range performance to approximately 128 dB.

NOTE: The ASRC can be programmed to add the triangular Probability Distribution Function (PDF) dither to the digital audio samples. It is advisable to add dither when the input word width exceeds the output word width, for example the input word is 20 bits and the output word is 16 bits. Triangular PDF is generally considered to create the most favorable noise shaping of the residual quantization noise.

Muting Modes

The mute feature of the ASRC can be controlled automatically in hardware using the `MUTE_IN` signal by connecting it to the `MUTE_OUT` signal. Automatic muting can be disabled by setting (=1) the `ASRC_MUTE.MUTE0` through `ASRC_MUTE.MUTE3` bits.

NOTE: Note that by default, the `ASRC_MUTE` register connects the `MUTE_IN` signal to the `MUTE_OUT` signal, but not conversely.

Soft Mute

When the `ASRC_CTL01.SOFTMUTE0`, `ASRC_CTL01.SOFTMUTE1`, `ASRC_CTL23.SOFTMUTE2` and `ASRC_CTL23.SOFTMUTE3` bits are set, the `MUTE_IN` signal is asserted, and the ASRC performs a soft mute by linearly decreasing the input data to the ASRC FIFO to zero, (-144 dB) attenuation as described for automatic hardware muting.

A 12-bit counter, clocked by `SRCx_FS_IP_I`, is used to control the mute attenuation. Therefore, the time it takes from the assertion of the `MUTE_IN` signal to -144 dB, full mute attenuation is 4096 FS cycles. Likewise, the time it takes to reach 0 dB mute attenuation from the deassertion of the `MUTE_IN` signal is 4096 FS cycles.

Hard Mute

When the `ASRC_CTL01.HARDMUTE0`, `ASRC_CTL01.HARDMUTE1`, `ASRC_CTL23.HARDMUTE2` and `ASRC_CTL23.HARDMUTE3` bits are set, the ASRC immediately mutes the input data to the ASRC FIFO to zero, (-144 dB) attenuation.

Auto Mute

When the `ASRC_CTL01.AUTOMUTE0`, `ASRC_CTL01.AUTOMUTE1`, `ASRC_CTL23.AUTOMUTE2` and `ASRC_CTL23.AUTOMUTE3` bits are set, the ASRC communicates with the S/PDIF receiver peripheral to determine when the input should mute.

This mode is useful for automatic detection of non-PCM audio data received from the S/PDIF receiver.

Interrupts

The following sections provide information about interrupt sources, masking and servicing.

Sources

Each ASRC module drives one interrupt signal (mute out asserted). All these signals are connected into the `DAI_IRPTL_H` or `DAI_IRPTL_L` latch registers. The ASRC ports generate interrupts as described below.

SRC Mute Out

The SRC mute out signal can be used to generate interrupts on their rising edge, falling edge, or both, depending on how the DAI interrupt mask registers (`DAI_IMSK_RE/DAI_IMSK_FE`) are programmed. This programming allows the generation of `DAI_IRPTL_H/DAI_IRPTL_L` interrupts either entering mute, exiting muting or both. The `SRCx_MUTE_OUT` interrupt is generated only once when the SRC is locked (after 4096 FS input samples) and after changes to the sample ratio. Hard mute, soft mute, and auto mute only control the muting of the input data to the SRC.

Masking

The `DAI_IMSK_FE`, `DAI_IMSK_RE`, and `DAI_IMSK_PRI` registers must be unmasked accordingly. The `DAI_IRQH` and `DAI_IRQL` signals are routed to the system event controller (SEC).

Service

The ISR reads the `DAI_IRPTL_H` and `DAI_IRPTL_L` registers to clear the interrupt request.

Programming Model

The following is basic information on programming the ASRC module.

1. Program the `ASRC_CTL01` and `ASRC_CTL23` registers and keep the `ASRC_CTL01.EN0` through `ASRC_CTL23.EN3` bits cleared.
2. Set the `ASRC_CTL01.EN0` through `ASRC_CTL23.EN3` bits. After 4096 input port FS cycles the ASRC has un-muted.

Debug Features

The asynchronous sample rate converter allow the bypass mode. When the `ASRC_CTL01.BYP0` through `ASRC_CTL23.BYP3` bits are set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. This mode can be used for testing both ports when the input and output sample rates are at the same frequency, therefore both input and output ports can be routed to the same serial clock and frame sync.

ADSP-2156x ASRC Register Descriptions

Sample Rate Converter Module (ASRC) contains the following registers.

Table 25-3: ADSP-2156x ASRC Register List

Name	Description
ASRC_CTL01	Control Register for ASRC 0 and 1
ASRC_CTL23	Control Register for ASRC 2 and 3
ASRC_MUTE	Mute Register
ASRC_RAT01	Ratio Register for ASRC 0 and 1
ASRC_RAT23	Ratio Register for ASRC 2 and 3

Control Register for ASRC 0 and 1

The `ASRC_CTL01` register (read/write) controls the operating modes, filters, and data formats used in the ASRC modules 0 and 1.

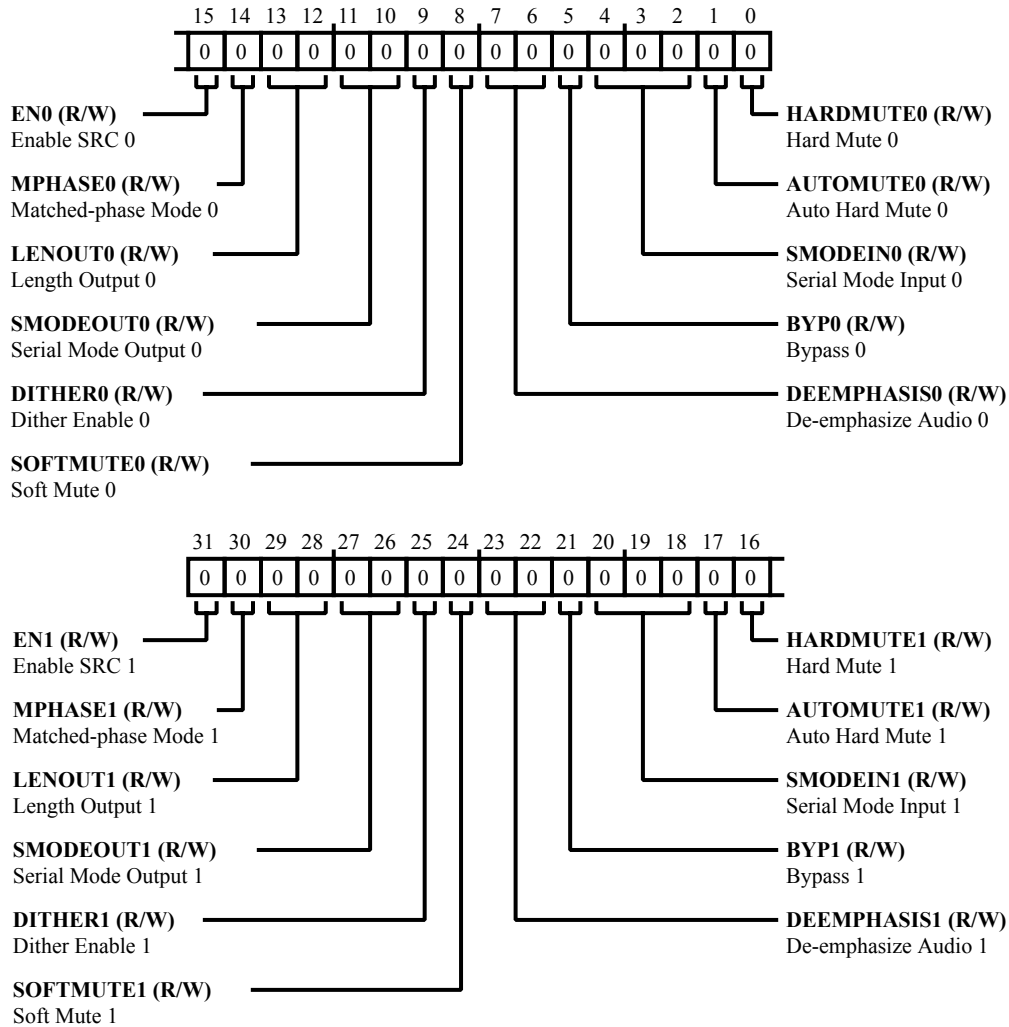


Figure 25-6: ASRC_CTL01 Register Diagram

Table 25-4: ASRC_CTL01 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration								
31 (R/W)	EN1	<p>Enable SRC 1.</p> <p>The ASRC_CTL01.EN1 bit enables SRC 1. When (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB).</p> <p>Note that SRC power-up completion is finished by clearing the ASRC_RAT01.MUTEOUT1 bit.</p> <p>Writes to the ASRC_CTL01 register should be at least one cycle before setting the ASRC_CTL01.EN1 bit. When setting and clearing this bit, it should be held low for a minimum of 5 SCLK cycles.</p>								
30 (R/W)	MPHASE1	<p>Matched-phase Mode 1.</p> <p>The ASRC_CTL01.MPHASE1 bit configures SRC1 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.</p> <table border="1"> <tr> <td>0</td> <td>Matched phase slave disabled</td> </tr> <tr> <td>1</td> <td>Matched phase slave enabled</td> </tr> </table>	0	Matched phase slave disabled	1	Matched phase slave enabled				
0	Matched phase slave disabled									
1	Matched phase slave enabled									
29:28 (R/W)	LENOUT1	<p>Length Output 1.</p> <p>The ASRC_CTL01.LENOUT1 bit field selects the serial output word length on SRC1.</p> <table border="1"> <tr> <td>0</td> <td>24 bits</td> </tr> <tr> <td>1</td> <td>20 bits</td> </tr> <tr> <td>2</td> <td>18 bits</td> </tr> <tr> <td>3</td> <td>16 bits</td> </tr> </table>	0	24 bits	1	20 bits	2	18 bits	3	16 bits
0	24 bits									
1	20 bits									
2	18 bits									
3	16 bits									
27:26 (R/W)	SMODEOUT1	<p>Serial Mode Output 1.</p> <p>The ASRC_CTL01.SMODEOUT1 bit field selects the serial output format on SRC1.</p> <table border="1"> <tr> <td>0</td> <td>Left-justified</td> </tr> <tr> <td>1</td> <td>I2S</td> </tr> <tr> <td>2</td> <td>TDM</td> </tr> <tr> <td>3</td> <td>Right-justified</td> </tr> </table>	0	Left-justified	1	I2S	2	TDM	3	Right-justified
0	Left-justified									
1	I2S									
2	TDM									
3	Right-justified									
25 (R/W)	DITHER1	<p>Dither Enable 1.</p> <p>The ASRC_CTL01.DITHER1 bit enables dithering before truncation on SRC1 when a word length less than 24 bits is selected.</p> <table border="1"> <tr> <td>0</td> <td>Truncation only</td> </tr> <tr> <td>1</td> <td>Dithering before truncation</td> </tr> </table>	0	Truncation only	1	Dithering before truncation				
0	Truncation only									
1	Dithering before truncation									

Table 25-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	SOFTMUTE1	Soft Mute 1. The ASRC_CTL01.SOFTMUTE1 bit enables soft mute on SRC1.
		0 Unmute
		1 Mute
23:22 (R/W)	DEEMPHASIS1	De-emphasize Audio 1. The ASRC_CTL01.DEEMPHASIS1 bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).
		0 No de-emphasis
		1 32 kHz
		2 44.1 kHz
		3 48 kHz
21 (R/W)	BYP1	Bypass 1. The ASRC_CTL01.BYP1 bit makes the output of SRC1 the same as the input.
20:18 (R/W)	SMODEIN1	Serial Mode Input 1. The ASRC_CTL01.SMODEIN1 bit field selects the serial input format for SRC1.
		0 left-justified
		1 I2S
		2 TDM
		4 24-bit right-justified
		5 20-bit right-justified
		6 18-bit right-justified
		7 16-bit right-justified
17 (R/W)	AUTOMUTE1	Auto Hard Mute 1. The ASRC_CTL01.AUTOMUTE1 bit auto hard mutes SRC1 when non audio is asserted by the SPDIF receiver.
		0 Unmute
		1 Mute
16 (R/W)	HARDMUTE1	Hard Mute 1. The ASRC_CTL01.HARDMUTE1 bit hard mutes SRC1.
		0 Unmute
		1 Mute

Table 25-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration								
15 (R/W)	EN0	<p>Enable SRC 0.</p> <p>The ASRC_CTL01.EN0 bit enables SRC 0. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB).</p> <p>Note that SRC power-up completion is finished by clearing the ASRC_RATE0.MUTEOUT0 bit.</p> <p>Writes to the ASRC_CTL01 register should be at least one cycle before setting the ASRC_CTL01.EN0 bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.</p>								
14 (R/W)	MPHASE0	Matched-phase Mode 0.								
		The ASRC_CTL01.MPHASE0 bit configures SRC0 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.								
		<table border="1"> <tr> <td>0</td> <td>Matched phase slave disabled</td> </tr> <tr> <td>1</td> <td>Matched phase slave enabled</td> </tr> </table>	0	Matched phase slave disabled	1	Matched phase slave enabled				
0	Matched phase slave disabled									
1	Matched phase slave enabled									
13:12 (R/W)	LENOUT0	Length Output 0.								
		The ASRC_CTL01.LENOUT0 bit field selects the serial output word length on SRC0.								
		<table border="1"> <tr> <td>0</td> <td>24 bits</td> </tr> <tr> <td>1</td> <td>20 bits</td> </tr> <tr> <td>2</td> <td>18 bits</td> </tr> <tr> <td>3</td> <td>16 bits</td> </tr> </table>	0	24 bits	1	20 bits	2	18 bits	3	16 bits
		0	24 bits							
		1	20 bits							
2	18 bits									
3	16 bits									
0	24 bits									
1	20 bits									
11:10 (R/W)	SMODEOUT0	Serial Mode Output 0.								
		The ASRC_CTL01.SMODEOUT0 bit field selects the serial output format on SRC0.								
		<table border="1"> <tr> <td>0</td> <td>Left-justified</td> </tr> <tr> <td>1</td> <td>I2S</td> </tr> <tr> <td>2</td> <td>TDM</td> </tr> <tr> <td>3</td> <td>Right-justified</td> </tr> </table>	0	Left-justified	1	I2S	2	TDM	3	Right-justified
		0	Left-justified							
		1	I2S							
2	TDM									
3	Right-justified									
0	Left-justified									
1	I2S									
9 (R/W)	DITHER0	Dither Enable 0.								
		The ASRC_CTL01.DITHER0 bit enables dithering before truncation on SRC0 when a word length less than 24 bits is selected.								
		<table border="1"> <tr> <td>0</td> <td>Truncation only</td> </tr> <tr> <td>1</td> <td>Dithering before truncation</td> </tr> </table>	0	Truncation only	1	Dithering before truncation				
0	Truncation only									
1	Dithering before truncation									

Table 25-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	SOFTMUTE0	Soft Mute 0. The ASRC_CTL01 . SOFTMUTE0 bit enables soft mute on SRC0.
		0 Unmute
		1 Mute
7:6 (R/W)	DEEMPHASIS0	De-emphasize Audio 0. The ASRC_CTL01 . DEEMPHASIS0 bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).
		0 No de-emphasis
		1 32 kHz
		2 44.1 kHz
		3 48 kHz
5 (R/W)	BYP0	Bypass 0. The ASRC_CTL01 . BYP0 bit makes the output of SRC0 the same as the input.
4:2 (R/W)	SMODEIN0	Serial Mode Input 0. The ASRC_CTL01 . SMODEIN0 bit field selects the serial input format for SRC0.
		0 left-justified
		1 I2S
		2 TDM
		4 24-bit right-justified
		5 20-bit right-justified
		6 18-bit right-justified
		7 16-bit right-justified
1 (R/W)	AUTOMUTE0	Auto Hard Mute 0. The ASRC_CTL01 . AUTOMUTE0 bit auto hard mutes SRC0 when non audio is asserted by the SPDIF receiver.
		0 Unmute
		1 Mute
0 (R/W)	HARDMUTE0	Hard Mute 0. The ASRC_CTL01 . HARDMUTE0 bit hard mutes SRC0.
		0 Unmute
		1 Mute (default)

Control Register for ASRC 2 and 3

The `ASRC_CTL23` register (read/write) controls the operating modes, filters, and data formats used in the sample rate converter modules 2 and 3.

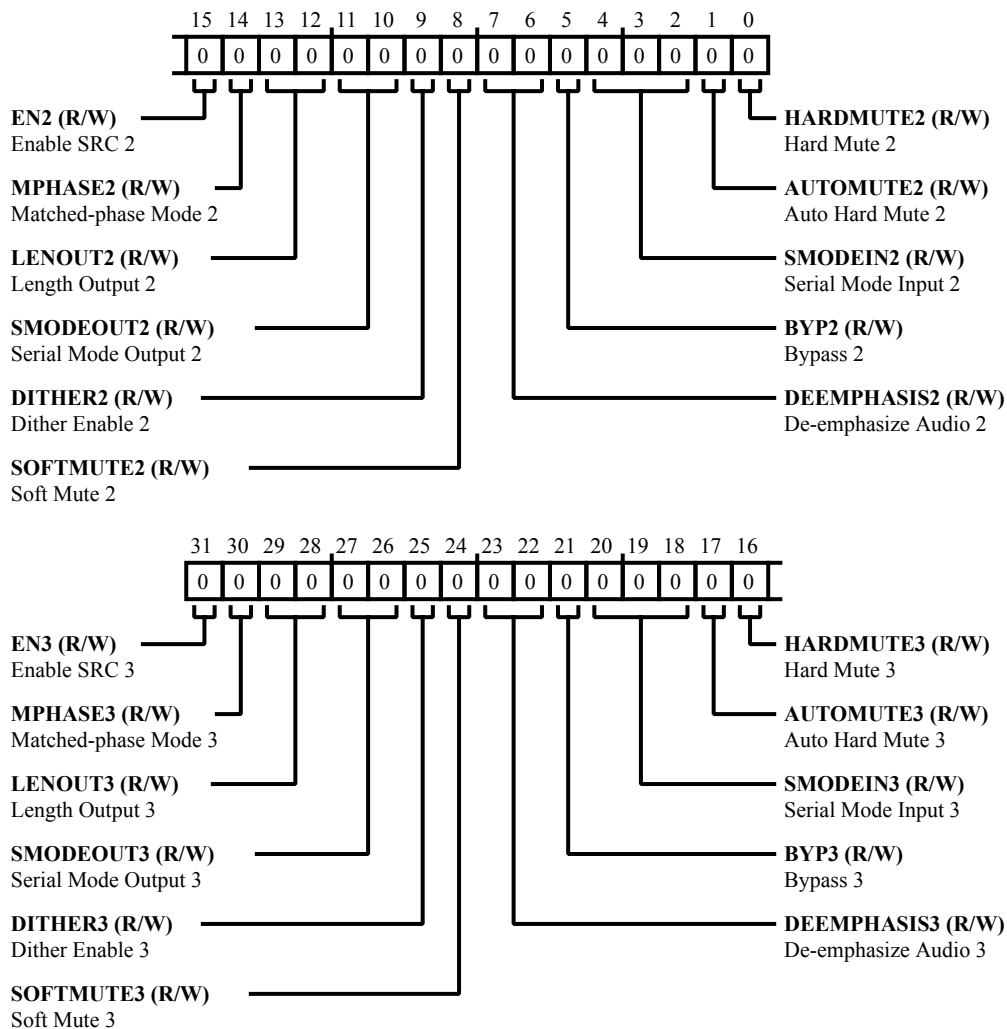


Figure 25-7: ASRC_CTL23 Register Diagram

Table 25-5: ASRC_CTL23 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration								
31 (R/W)	EN3	<p>Enable SRC 3.</p> <p>The ASRC_CTL23.EN3 bit enables SRC 3. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB).</p> <p>Note that SRC power-up completion is finished by clearing the ASRC_RAT23.MUTEOUT3 bit.</p> <p>Writes to the ASRC_CTL23 register should be at least one cycle before setting the ASRC_CTL23.EN3 bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.</p>								
30 (R/W)	MPHASE3	<p>Matched-phase Mode 3.</p> <p>The ASRC_CTL23.MPHASE3 bit configures SRC3 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.</p> <table border="1"> <tr> <td>0</td> <td>Matched phase slave disabled</td> </tr> <tr> <td>1</td> <td>Matched phase slave enabled</td> </tr> </table>	0	Matched phase slave disabled	1	Matched phase slave enabled				
0	Matched phase slave disabled									
1	Matched phase slave enabled									
29:28 (R/W)	LENOUT3	<p>Length Output 3.</p> <p>The ASRC_CTL23.LENOUT3 bit field selects the serial output word length on SRC3.</p> <table border="1"> <tr> <td>0</td> <td>24 bits</td> </tr> <tr> <td>1</td> <td>20 bits</td> </tr> <tr> <td>2</td> <td>18 bits</td> </tr> <tr> <td>3</td> <td>16 bits</td> </tr> </table>	0	24 bits	1	20 bits	2	18 bits	3	16 bits
0	24 bits									
1	20 bits									
2	18 bits									
3	16 bits									
27:26 (R/W)	SMODEOUT3	<p>Serial Mode Output 3.</p> <p>The ASRC_CTL23.SMODEOUT3 bit field selects the serial output format on SRC3.</p> <table border="1"> <tr> <td>0</td> <td>Left-justified</td> </tr> <tr> <td>1</td> <td>I2S</td> </tr> <tr> <td>2</td> <td>TDM</td> </tr> <tr> <td>3</td> <td>Right-justified</td> </tr> </table>	0	Left-justified	1	I2S	2	TDM	3	Right-justified
0	Left-justified									
1	I2S									
2	TDM									
3	Right-justified									
25 (R/W)	DITHER3	<p>Dither Enable 3.</p> <p>The ASRC_CTL23.DITHER3 bit enables dithering before truncation on SRC3 when a word length less than 24 bits is selected.</p> <table border="1"> <tr> <td>0</td> <td>Truncation only</td> </tr> <tr> <td>1</td> <td>Dithering before truncation</td> </tr> </table>	0	Truncation only	1	Dithering before truncation				
0	Truncation only									
1	Dithering before truncation									

Table 25-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	SOFTMUTE3	Soft Mute 3. The ASRC_CTL23.SOFTMUTE3 bit enables soft mute on SRC3.
		0 Unmute
		1 Mute
23:22 (R/W)	DEEMPHASIS3	De-emphasize Audio 3. The ASRC_CTL23.DEEMPHASIS3 bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).
		0 No de-emphasis
		1 32 kHz
		2 44.1 kHz
		3 48 kHz
21 (R/W)	BYP3	Bypass 3. The ASRC_CTL23.BYP3 bit makes the output of SRC3 the same as the input.
20:18 (R/W)	SMODEIN3	Serial Mode Input 3. The ASRC_CTL23.SMODEIN3 bit field selects the serial input format for SRC3.
		0 left-justified
		1 I2S
		2 TDM
		4 24-bit right-justified
		5 20-bit right-justified
		6 18-bit right-justified
		7 16-bit right-justified
17 (R/W)	AUTOMUTE3	Auto Hard Mute 3. The ASRC_CTL23.AUTOMUTE3 bit auto hard mutes SRC3 when non audio is asserted by the SPDIF receiver.
		0 Unmute
		1 Mute
16 (R/W)	HARDMUTE3	Hard Mute 3. The ASRC_CTL23.HARDMUTE3 bit hard mutes SRC3.
		0 Unmute
		1 Mute

Table 25-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration								
15 (R/W)	EN2	<p>Enable SRC 2.</p> <p>The <code>ASRC_CTL23.EN2</code> bit enables SRC 2. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) <code>MUTE_OUT</code> is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB).</p> <p>Note that SRC power-up completion is finished by clearing the <code>ASRC_RAT23.MUTEOUT2</code> bit.</p> <p>Writes to the <code>ASRC_CTL23</code> register should be at least one cycle before setting the <code>ASRC_CTL23.EN2</code> bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.</p>								
14 (R/W)	MPHASE2	<p>Matched-phase Mode 2.</p> <p>The <code>ASRC_CTL23.MPHASE2</code> bit configures SRC2 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.</p> <table border="1"> <tr> <td>0</td> <td>Matched phase slave disabled</td> </tr> <tr> <td>1</td> <td>Matched phase slave enabled</td> </tr> </table>	0	Matched phase slave disabled	1	Matched phase slave enabled				
0	Matched phase slave disabled									
1	Matched phase slave enabled									
13:12 (R/W)	LENOUT2	<p>Length Output 2.</p> <p>The <code>ASRC_CTL23.LENOUT2</code> bit field selects the serial output word length on SRC2.</p> <table border="1"> <tr> <td>0</td> <td>24 bits</td> </tr> <tr> <td>1</td> <td>20 bits</td> </tr> <tr> <td>2</td> <td>18 bits</td> </tr> <tr> <td>3</td> <td>16 bits</td> </tr> </table>	0	24 bits	1	20 bits	2	18 bits	3	16 bits
0	24 bits									
1	20 bits									
2	18 bits									
3	16 bits									
11:10 (R/W)	SMODEOUT2	<p>Serial Mode Output 2.</p> <p>The <code>ASRC_CTL23.SMODEOUT2</code> bit field selects the serial output format on SRC2.</p> <table border="1"> <tr> <td>0</td> <td>Left-justified</td> </tr> <tr> <td>1</td> <td>I2S</td> </tr> <tr> <td>2</td> <td>TDM</td> </tr> <tr> <td>3</td> <td>Right-justified</td> </tr> </table>	0	Left-justified	1	I2S	2	TDM	3	Right-justified
0	Left-justified									
1	I2S									
2	TDM									
3	Right-justified									
9 (R/W)	DITHER2	<p>Dither Enable 2.</p> <p>The <code>ASRC_CTL23.DITHER2</code> bit enables dithering before truncation on SRC2 when a word length less than 24 bits is selected.</p> <table border="1"> <tr> <td>0</td> <td>Truncation only</td> </tr> <tr> <td>1</td> <td>Dithering before truncation</td> </tr> </table>	0	Truncation only	1	Dithering before truncation				
0	Truncation only									
1	Dithering before truncation									

Table 25-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	SOFTMUTE2	Soft Mute 2. The ASRC_CTL23.SOFTMUTE2 bit enables soft mute on SRC2.
		0 Unmute
		1 Mute
7:6 (R/W)	DEEMPHASIS2	De-emphasize Audio 2. The ASRC_CTL23.DEEMPHASIS2 bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).
		0 No de-emphasis
		1 32 kHz
		2 44.1 kHz
		3 48 kHz
5 (R/W)	BYP2	Bypass 2. The ASRC_CTL23.BYP2 bit makes the output of SRC2 the same as the input.
4:2 (R/W)	SMODEIN2	Serial Mode Input 2. The ASRC_CTL23.SMODEIN2 bit field selects the serial input format for SRC2.
		0 left-justified
		1 I2S
		2 TDM
		4 24-bit right-justified
		5 20-bit right-justified
		6 18-bit right-justified
		7 16-bit right-justified
1 (R/W)	AUTOMUTE2	Auto Hard Mute 2. The ASRC_CTL23.AUTOMUTE2 bit auto hard mutes SRC2 when non audio is asserted by the SPDIF receiver.
		0 Unmute
		1 Mute
0 (R/W)	HARDMUTE2	Hard Mute 2. The ASRC_CTL23.HARDMUTE2 bit hard mutes SRC2.
		0 Unmute
		1 Mute

Mute Register

This register connects an ASRCx mute input and output when the mute bit is cleared (=0). This allows ASRCx to automatically mute input while the ASRC is initializing (0 = automatic muting and 1 = manual muting). Bit 0 controls ASRC0, bit 1 controls ASRC1, bit 2 controls ASRC2, and bit 3 controls ASRC3.

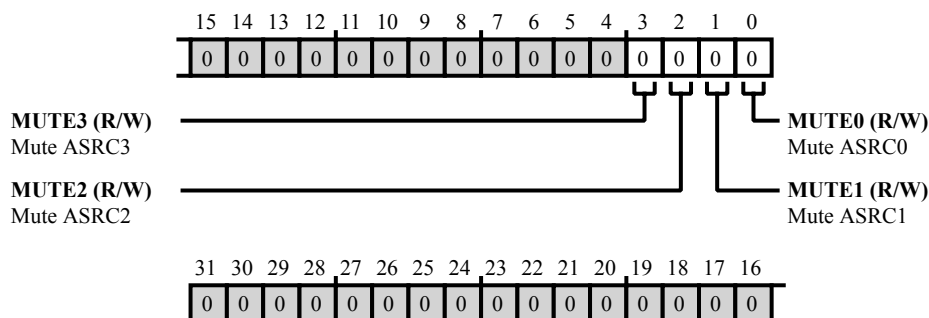


Figure 25-8: ASRC_MUTE Register Diagram

Table 25-6: ASRC_MUTE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	MUTE3	Mute ASRC3. The ASRC_MUTE.MUTE3 bit automatically mutes ASRC3 output when cleared (=0).
2 (R/W)	MUTE2	Mute ASRC2. The ASRC_MUTE.MUTE2 bit automatically mutes ASRC2 output when cleared (=0).
1 (R/W)	MUTE1	Mute ASRC1. The ASRC_MUTE.MUTE1 bit automatically mutes ASRC1 output when cleared (=0).
0 (R/W)	MUTE0	Mute ASRC0. The ASRC_MUTE.MUTE0 bit automatically mutes ASRC0 output when cleared (=0).

Ratio Register for ASRC 0 and 1

The `ASRC_RAT01` register report the mute and I/O sample ratio for ASRC0 and ASRC1.

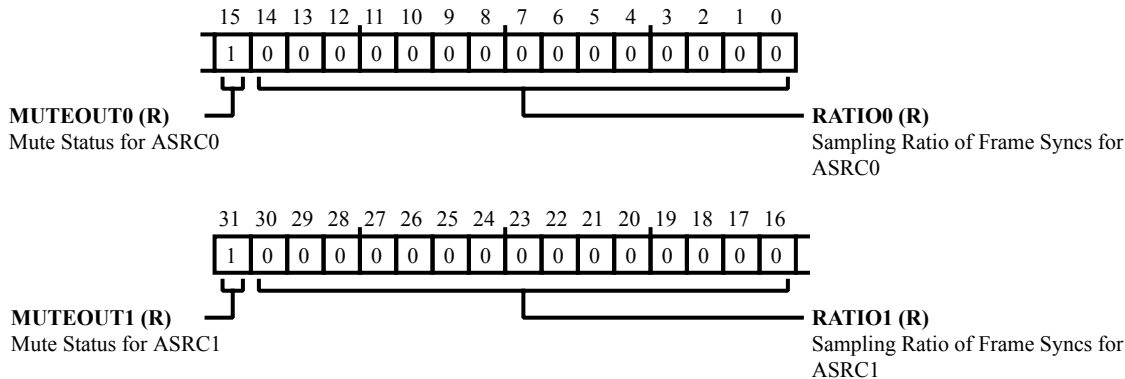


Figure 25-9: ASRC_RAT01 Register Diagram

Table 25-7: ASRC_RAT01 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MUTEOUT1	Mute Status for ASRC1. The <code>ASRC_RAT01.MUTEOUT1</code> bit field reports the status of the <code>MUTE_OUT</code> signal. Once the <code>SRCx_MUTEOUT</code> signal is cleared, the ratio can be read. When ASRC1 is enabled or there is a change in the sample ratio, the <code>MUTE_OUT</code> signal is asserted. The <code>MUTE_OUT</code> signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the <code>MUTE_OUT</code> signal is deasserted.
30:16 (R/NW)	RATIO1	Sampling Ratio of Frame Syncs for ASRC1. The <code>ASRC_RAT01.RATIO1</code> bit field is read to find the ratio of output to input sampling frequency for ASRC1 (<code>SRCx_FS_OP_I/SRCx_FS_IP_I</code>). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15 (R/NW)	MUTEOUT0	Mute Status for ASRC0. The <code>ASRC_RAT01.MUTEOUT0</code> bit field reports the status of the <code>MUTE_OUT</code> signal. Once the <code>SRCx_MUTEOUT</code> signal is cleared, the ratio can be read. When ASRC0 is enabled or there is a change in the sample ratio, the <code>MUTE_OUT</code> signal is asserted. The <code>MUTE_OUT</code> signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the <code>MUTE_OUT</code> signal is deasserted.

Table 25-7: ASRC_RAT01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	RATIO0	<p>Sampling Ratio of Frame Syncs for ASRC0.</p> <p>The <code>ASRC_RAT01.RATIO0</code> bit field is read to find the ratio of output to input sampling frequency for ASRC0 (<code>SRCx_FS_OP_I/SRCx_FS_IP_I</code>). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.</p>

Ratio Register for ASRC 2 and 3

The `ASRC_RAT23` register report the mute and I/O sample ratio for ASRC0 and ASRC1.

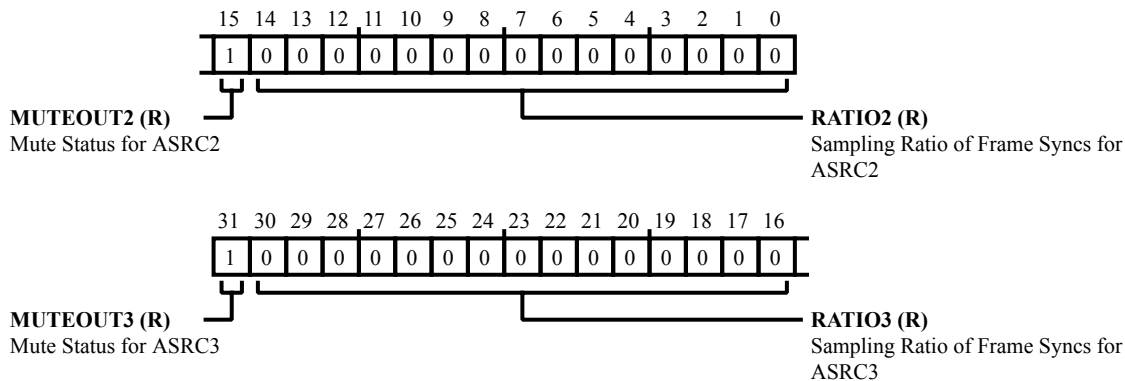


Figure 25-10: ASRC_RAT23 Register Diagram

Table 25-8: ASRC_RAT23 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MUTEOUT3	Mute Status for ASRC3. The <code>ASRC_RAT23.MUTEOUT3</code> bit field reports the status of the <code>MUTE_OUT</code> signal. Once the <code>SRCx_MUTEOUT</code> signal is cleared, the ratio can be read. When ASRC3 is enabled or there is a change in the sample ratio, the <code>MUTE_OUT</code> signal is asserted. The <code>MUTE_OUT</code> signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the <code>MUTE_OUT</code> signal is deasserted.
30:16 (R/NW)	RATIO3	Sampling Ratio of Frame Syncs for ASRC3. The <code>ASRC_RAT23.RATIO3</code> bit field is read to find the ratio of output to input sampling frequency for ASRC3 (<code>SRCx_FS_OP_I/SRCx_FS_IP_I</code>). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15 (R/NW)	MUTEOUT2	Mute Status for ASRC2. The <code>ASRC_RAT23.MUTEOUT2</code> bit field reports the status of the <code>MUTE_OUT</code> signal. Once the <code>SRCx_MUTEOUT</code> signal is cleared, the ratio can be read. When ASRC2 is enabled or there is a change in the sample ratio, the <code>MUTE_OUT</code> signal is asserted. The <code>MUTE_OUT</code> signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the <code>MUTE_OUT</code> signal is deasserted.

Table 25-8: ASRC_RAT23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	RATIO2	<p>Sampling Ratio of Frame Syncs for ASRC2.</p> <p>The <code>ASRC_RAT23.RATIO2</code> bit field is read to find the ratio of output to input sampling frequency for ASRC2 (<code>SRCx_FS_OP_I/SRCx_FS_IP_I</code>). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.</p>

26 Sony/Philips Digital Interface (S/PDIF)

The Sony/Philips Digital Interface (S/PDIF) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. The digital audio interface carries three types of information; audio data, non audio data (compressed data) and timing information.

Features

The S/PDIF interface has the following features.

- Supports one stereo channel or compressed audio streams.
- AES3-compatible S/PDIF transmitter and receiver.
- Transmitting a biphase mark encoded signal that may contain any number of audio channels (compressed or linear pulse code modulation) or non-audio data.
- S/PDIF receiver managing clock recovery with separate S/PDIF on-chip PLL.
- S/PDIF receiver supports the detection of DTS frames of 256, 512, 1024, 2048, and 4096.
- Manage user status information and provide error-handling capabilities in both the transmitter and receiver.
- DAI allows interactions over DAI by serial ports and the external DAI pins to interface to other S/PDIF devices. This includes using the receiver to decode incoming biphase encoded audio streams and passing them via the SPORTs to internal memory for processing-or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system.

It is important to be familiar with serial digital audio interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

ADSP-2156x SPDIF Register List

The S/PDIF module is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. A set of registers govern S/PDIF operations. For more information on S/PDIF functionality, see the S/PDIF register descriptions.

Table 26-1: ADSP-2156x SPDIF Register List

Name	Description
SPDIF_RX_CTL	Receive Control
SPDIF_RX_STAT	Receive Status Register
SPDIF_RX_STAT0_A	Receive Status A0 Register
SPDIF_RX_STAT0_B	Receive Status B0 Register
SPDIF_RX_STAT1_A	Receive Status A1 Register
SPDIF_RX_STAT1_B	Receive Status B1 Register
SPDIF_TX_CTL	Transmit Control Register
SPDIF_TX_STAT_A0	Transmit Status A0 Register
SPDIF_TX_STAT_A1	Transmit Status A1 Register
SPDIF_TX_STAT_A2	Transmit Status A2 Register
SPDIF_TX_STAT_A3	Transmit Status A3 Register
SPDIF_TX_STAT_A4	Transmit Status A4 Register
SPDIF_TX_STAT_A5	Transmit Status A5 Register
SPDIF_TX_STAT_B0	Transmit Status B0 Register
SPDIF_TX_STAT_B1	Transmit Status B1 Register
SPDIF_TX_STAT_B2	Transmit Status B2 Register
SPDIF_TX_STAT_B3	Transmit Status B3 Register
SPDIF_TX_STAT_B4	Transmit Status B4 Register
SPDIF_TX_STAT_B5	Transmit Status B5 Register
SPDIF_TX_UBUFF_A0	Transmit User Buffer A0 Register
SPDIF_TX_UBUFF_A1	Transmit User Buffer A1 Register
SPDIF_TX_UBUFF_A2	Transmit User Buffer A2 Register
SPDIF_TX_UBUFF_A3	Transmit User Buffer A3 Register
SPDIF_TX_UBUFF_A4	Transmit User Buffer A4 Register
SPDIF_TX_UBUFF_A5	Transmit User Buffer A5 Register
SPDIF_TX_UBUFF_B0	Transmit User Buffer B0 Register
SPDIF_TX_UBUFF_B1	Transmit User Buffer B1 Register
SPDIF_TX_UBUFF_B2	Transmit User Buffer B2 Register
SPDIF_TX_UBUFF_B3	Transmit User Buffer B3 Register
SPDIF_TX_UBUFF_B4	Transmit User Buffer B4 Register
SPDIF_TX_UBUFF_B5	Transmit User Buffer B5 Register

Table 26-1: ADSP-2156x SPDIF Register List (Continued)

Name	Description
SPDIF_TX_USRUPDT	User Bit Update Register

SRU Programming

The SRU (signal routing unit) is used to connect the S/PDIF transmitter biphasic data out to the output pins or to the S/PDIF receiver. The serial clock, frame sync, data, and external sync (if external synchronization is required) inputs also need to be routed through the SRU. For details of the routing, see the [DAI Routing Capabilities](#) section in the *Digital Audio Interface (DAI)* chapter.

The SRU needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the connection to the input biphasic stream.

Program the corresponding SRU registers to connect the outputs to the required destinations (see the [DAI Routing Capabilities](#) section). The biphasic encoded data and the external PLL clock inputs to the receiver are routed through the SRU. The extracted clock, frame sync, and data are also routed through the SRU.

S/PDIF Interrupt List

Table 26-2: S/PDIF Interrupt List

Interrupt Name	Interrupt Condition	Return DAI Register	Return SEC Register	SEC ID
DAI0_IRQH	Block Start	DAIx_IRPTL	SEC_ID	24
DAI0_IRQL	Non Audio			145

Clocking

The fundamental timing clock of the S/PDIF receiver is SCLK1_0 from the CDU. When SCLK1_0 is configured, it supports sampling frequencies of 32 kHz to 192 kHz. The fundamental timing clock of the S/PDIF transmitter is *SCLK0/4*.

For information on clock programming, see [CDU Programming Model](#).

S/PDIF Transmitter

The following sections provide information on the S/PDIF transmitter.

Functional Description

The S/PDIF transmitter, shown in the *S/PDIF Transmitter Block Diagram*, resides within the DAI, and its inputs and outputs can be routed via the SRU. It receives audio data in serial format, encloses the specified user status information, and converts it into the biphasic encoded signal. The serial data input to the transmitter can be

formatted as left-justified, I²S, or right-justified with word widths of 16, 18, 20 or 24 bits. *AES3 Output Block* shows the detail of the AES block.

The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the signal routing unit (SRU).

The S/PDIF transmitter output may be routed to an output pin via the SRU and then routed to another S/PDIF receiver or to components for off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU.

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status/user registers for a frame (192-bits/24 bytes) in the transmitter that correspond to each channel or subframe.

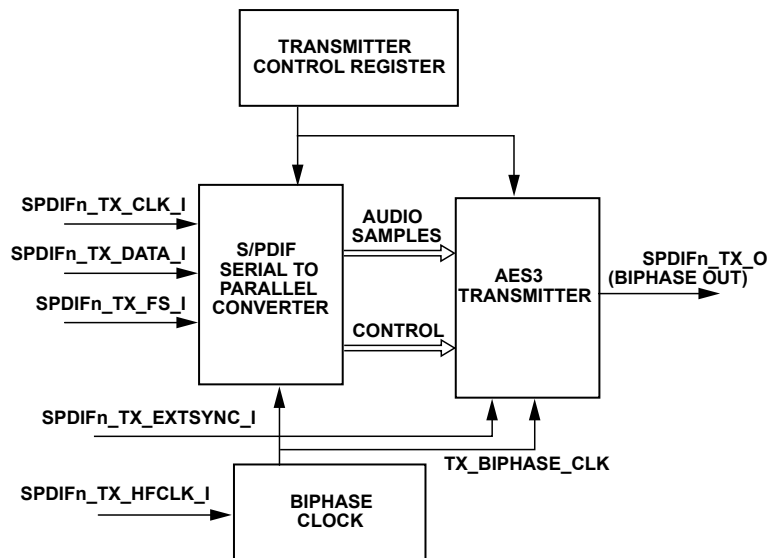


Figure 26-1: S/PDIF Transmitter Block Diagram

Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data.

A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I²S, or right-justified with 16-, 18-, 20- or 24-bit word widths. The over sampling clock is also selected by the transmitter control register.

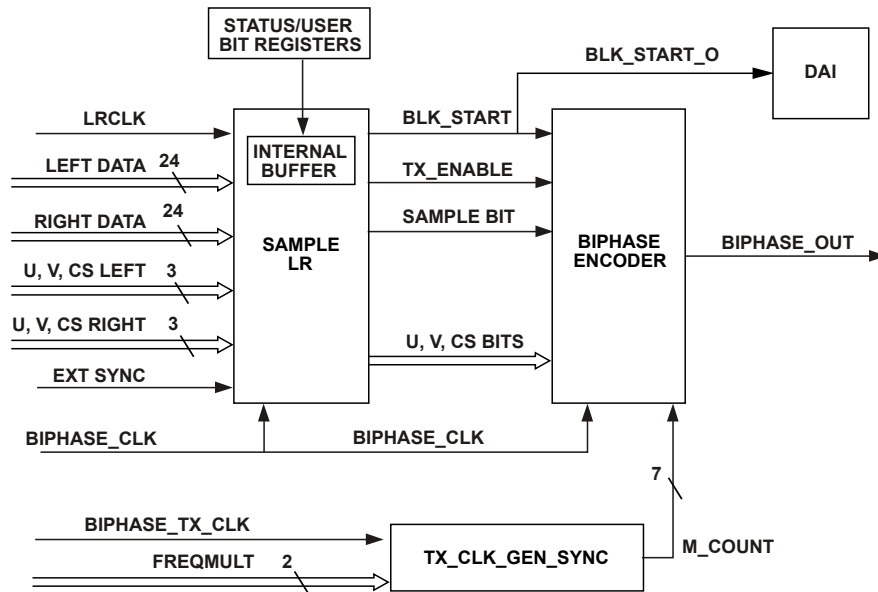


Figure 26-2: AES3 Output Block

Input Data Formats

The *I²S and Left-Justified Formats* and *Right-Justified Formats* figures show the format of data that is sent to the S/PDIF transmitter using a variety of protocol standards.

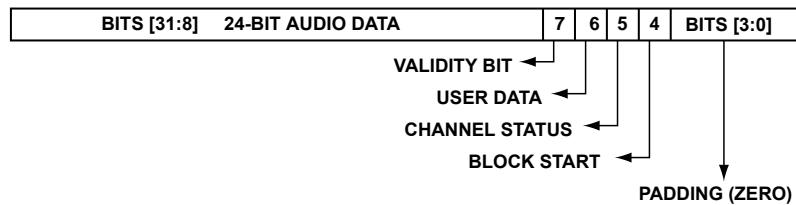


Figure 26-3: I²S and Left-Justified Formats

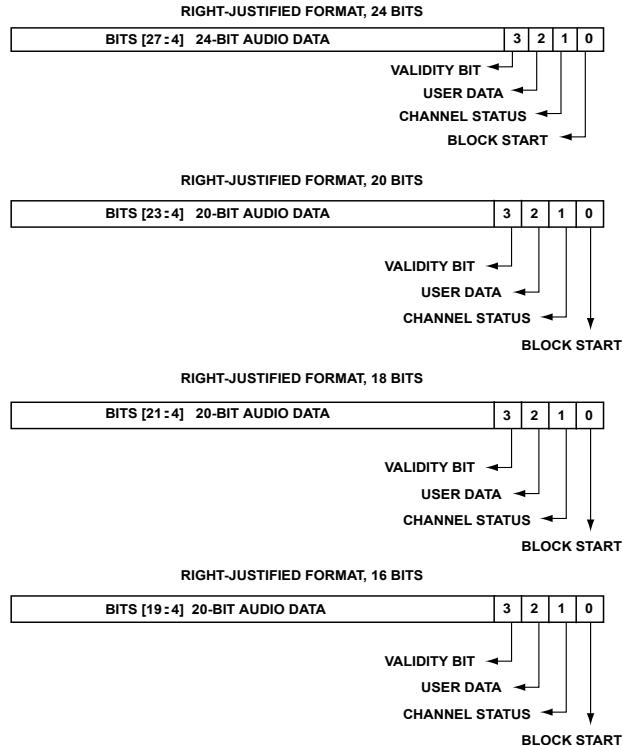


Figure 26-4: Right-Justified Formats

Operating Modes

The S/PDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Full Serial Mode

This mode is selected by clearing the `SPDIF_TX_CTL.AUTO` bit. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the serial data stream (`SPDIF_TX_DATA_I`) pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Standalone Mode

This mode is selected by setting the `SPDIF_TX_CTL.AUTO` bit. In this mode, the block start bit (indicating the start of a frame) is generated internally. The channel status bits come from the channel status buffer registers. The user status bits come from the user bits buffers as shown in the *AES3 Output Block* figure.

The validity bits are the `SPDIF_TX_CTL.VALIDR` and `SPDIF_TX_CTL.VALIDL`. In this mode only audio data comes from the `SPDIF_TX_DATA_I` pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Once the user bits buffer registers (`SPDIF_TX_UBUFF_A0` - `SPDIF_TX_UBUFF_B5`) are programmed, they are used only for the next block of data. This allows programs to change the user bit information in every block of data.

To allow user bit updates, write a 0x1 to the `SPDIF_TX_USRUPDT` register that is used for further processing. If the `SPDIF_TX_CTL.AUTO` bit is set:

- and if `SPDIF_TX_USRUPDT = 1`, at every 192nd frame end the user status bits are taken from user bits buffers and transmitted. Simultaneously, the `SPDIF_TX_USRUPDT` register is cleared automatically by hardware.
- and if `SPDIF_TX_USRUPDT = 0`, at every 192nd frame end the user status bits are updated as zeros and transmitted. The `SPDIF_TX_USRUPDT` register remains low.

In general, for the next block, programs can update user bits buffers at any time during the transfer of the current block (1 block = 192 frames). There are internal buffers to store the user status bits of the current block of transfer. In other words, at the beginning of every new block, the user status bit (`SPDIF_TX_CTL.USRPEND` bit) from user bits buffers are copied to internal buffers and transmitted in each frame during the transfer.

Note that since a frame contains $192 \text{ bits}/8 = 24$ bytes, six status/user registers are required to store each four bytes.

Data Output Mode

Two output data formats are supported by the transmitter; *two channel mode* and *single-channel double-frequency* (SCDF) mode. The output format is determined by the transmitter control register (`SPDIF_TX_CTL`).

In two channel mode, the left channel (channel A) is transmitted when the `SPDIF_TX_FS_I` is high and the right channel (channel B) is transmitted when the `SPDIF_TX_FS_I` is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both sub frames, instead of channel A and B. The transmitter will transmit at half the sample rate of the input bit stream. The `SPDIF_TX_CTL.SCDF` bit selects SCDF mode. When in SCDF mode, the `SPDIF_TX_CTL.SCDFLR` bit determines whether left or right channel data is transmitted.

S/PDIF Receiver

The S/PDIF receiver (*S/PDIF Receiver Block Diagram*) is compliant with all common serial digital audio interface standards including IEC-60958, IEC-61937, AES3, and AES11. For the IEC-60958 standard, all the user-data and channel-status bits (as outlined in this document) are not decoded by the S/PDIF receiver. The interface does make all 192 user-data and channel-status pairs available as an output of the block, for post-decoding.

For the IEC-61937 standard, the S/PDIF only detects compressed AC-3 and DTS formats. No decompression is performed.

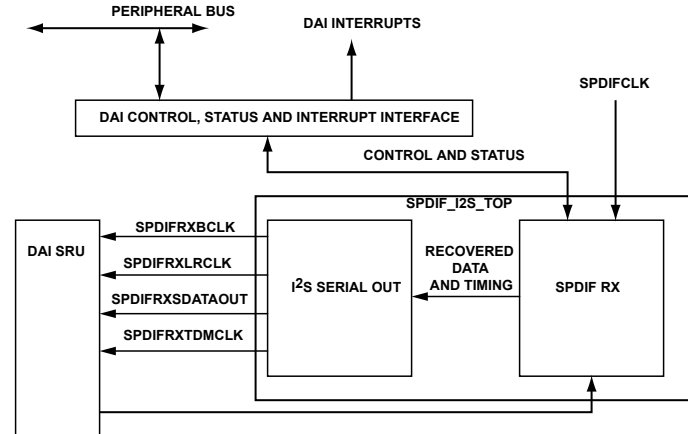


Figure 26-5: S/PDIF Receiver Block Diagram

Functional Description

If the receiver is used, programs need to enable it using the `SPDIF_RX_CTL` register. After the SRU programming is complete, write to the register with control values. At this point, the receiver attempts to lock.

NOTE: The S/PDIF receiver is disabled at default. If the receiver is used in an application, programs should enable the receiver.

The input to the receiver (`SPDIFn_RX_I`) is a biphasic encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single biphasic encoded stream, producing an I²S compatible serial data output that consists of a serial clock, a left-right frame sync, and data (channel A/B). It provides the programmer with several methods of managing the incoming status bit information.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 24 kHz to 192 kHz. Refer to [Clocking](#) for more details.

The channel status bits are collected into memory-mapped registers, while other channel status and user bytes must be handled manually. The block start bit, which replaces the parity bit in the serial I²S stream, indicates the reception of the Z preamble and the start of a new block of channel status and data bits.

Clock Recovery

The S/PDIF receiver recovers the clock that generated the AES3/SPDIF biphasic encoded stream from the incoming S/PDIF stream.

This clock is used by the receiver to clock in the biphasic encoded data stream and also to provide clocks for either the SPORTs, sample rate converter, or the AES3 and S/PDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

In order to maintain performance, jitter on the clock is sourced to several peripherals.

To comply with the AES11 standard, the recovered left or right clock must be aligned with the preambles within a + or - 5% of the frame period. Since the PLL clock generates a clock 512 times the frame rate clock ($512 f_{SCLK}$), this clock can be used and divided down to create the phase aligned jitter-free left or right clock.

TDM Clock Output

The SPDIF receiver generates the TDMCLK clock signal which is 256 times the sampling rate. For proper generation of TDMCLK, program the SPDIF receiver as shown in the *TDMCLK Configuration* table.

IMPORTANT: The SPDIF receiver guarantees 256 clocks in one frame sync. It does not guarantee the duty cycle of the generated clocks. Therefore, do not use this clock in the applications that have duty cycle requirements.

Table 26-3: TDMCLK Configuration

SPDIF_RX_CTL.TDMSEL	Recommended Sampling Rate
0000	$\leq 96K$
0001	192K
0011	96K - 192K
0101	96K

Output Data Format

The extracted 24-bit audio data, V, U, C and block start bits are sent on the SPDIF_RX_DAT pin in 32-bit I²S format as shown in *I2S and Left-Justified Formats*. The frame sync is transmitted on the SPDIF_RX_FS pin and serial clock is transmitted on the SPDIF_RX_CLK pin. All three pins are routed through the SRU.

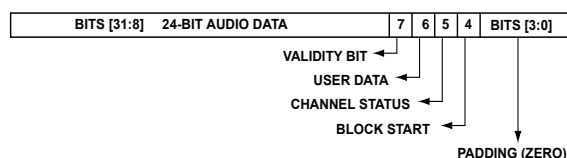


Figure 26-6: I2S Format

Channel Status

The channel status for the first bytes 4-0 (consumer mode) are collected into memory-mapped registers (`SPDIF_RX_STAT0_A`, `SPDIF_RX_STAT0_B`, `SPDIF_RX_STAT1_A` and `SPDIF_RX_STAT1_B`). All other channel status bytes 23-5 (professional mode) must be manually extracted from the receiver data stream.

NOTE: Only the first 5 channel status bytes (40-bit) for consumer mode of a frame are stored into the S/PDIF receiver status registers.

Operating Modes

This section describes the receiver channel status for the different modes.

Compressed or Non-linear Audio Data

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how this peripheral handles different data.

MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the `SPDIF_RX_STAT.VALID` bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of 0xF872 and another word consisting of 0x4E1F. When this sync code is detected, the `SPDIF_RX_STAT.COMPMODE` bits hold the information regarding type of compression.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.

Emphasized Audio Data

Determination as to whether the received audio data is emphasized or not is done in software using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2-4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.
- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3-5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

Single-Channel Double-Frequency Mode

Unlike previous processors, support for single-channel, double-frequency mode (SCDF) is not supported through specific bits within the `SPDIF_RX_CTL` register, but rather have to be implemented in software using the information provided by the CS (channel status) bits.

- 0111 = single channel double frequency mode
- 1000 = single channel double frequency mode-stereo left
- 1001 = single channel double frequency mode-stereo right

Clock Recovery Modes

The S/PDIF receiver extracts audio data, channel status, and user bits from the biphase encoded AES3 and S/PDIF stream. In addition, a 50% duty cycle reference clock running at the sampling rate of the audio input data is generated for the receiver to recover the oversampling clock.

Number Controlled Oscillator

The receiver can recover the clock from the biphase encoded stream using an on-chip NCO shown in the following figure. Note the dedicated NCO is separate from the PLL that supplies the clock to the processor core and which is the default operation of the receiver.

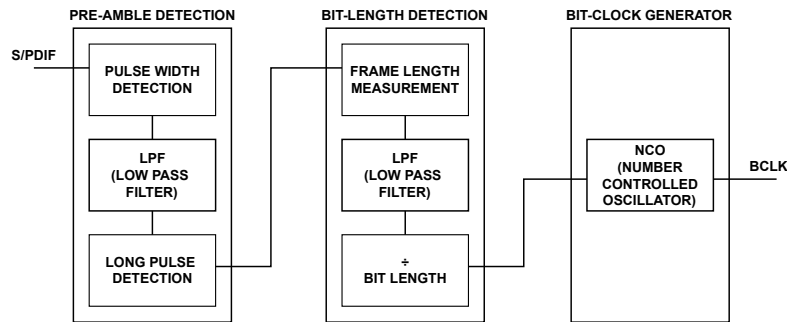


Figure 26-7: S/PDIF Clock Recovery Mechanism

The left/right frame reference clock for the NCO is generated using the preambles. The recovered low jitter left/right frame clock from the NCO attempts to align with the reference clock. However, this recovered left/right clock, like the reference clock, is not phase aligned with the preambles.

Interrupts

The following sections provide information about interrupt sources, masking and servicing.

Sources

The S/PDIF module of each DAI drives five interrupt signals. Four are status signals driven from SPDIFn_RX and one signal is driven from SPDIFn_TX (block start). These signals are connected into the `DAI_IRPTL_L/DAI_IRPTL_H` latch register.

Transmit Block Start

The `SPDIFn_TX_BLKSTART` output signal, if routed to any miscellaneous interrupt bits (`DAIn_INT_31-22` in the `DAI_MISCO/DAI_MISC1` registers), triggers a block start interrupt during the last frame of current block.

Receiver Status

The following receiver status bits generate an interrupt.

- Validity (`SPDIF_RX_STAT.VALID`)
- Receiver locked (`SPDIF_RX_STAT.LOCK`)
- No audio (`SPDIF_RX_STAT.AUDIOTYPE`)

Receiver Error

The loss of lock (`SPDIF_RX_STAT.LOCKLOSS`) bit generates an interrupt.

Masking

For the S/PDIF receiver the `DAI_IMSK_RE` register must be unmasked accordingly. For the S/PDIF transmit the `DAIn_IMASK_x` register must be unmasked accordingly.

The `INTR_DAI_IRQH` and `INTR_DAI_IRQL` signals are routed to the SEC.

Service

The ISR reads the `DAI_IRPTL_H` and `DAI_IRPTL_L` registers to clear the interrupt request.

Programming Model

The following sections provide information on programming the transmitter and receiver.

Programming the Transmitter

Since the S/PDIF transmitter data input is not available to the core, programming the transmitter is as simple as: 1) connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be encoded, and 2) selecting the desired mode in the transmitter control register. This setup can be accomplished in three steps.

1. Connect the transmitter's four required input signals and one biphase encoded output in the SRU. The four input signals are the serial clock (`SPDIF_TX_CLK_I`), the serial frame sync (`SPDIF_TX_FS_I`), the serial data (`SPDIF_TX_DAT_I`), and the high frequency clock (`SPDIF_TX_HFCLK_I`) used for the encoding. The only output of the transmitter is `SPDIF_TX_O`.
2. If user bits are required, write 0x1 to the `SPDIF_TX_USRUPDT` register for the first block of transfer. Also route the `SPDIF_TX_BLK_START_O` signal to the `DAI_INT_31-22` (`DAI_IRPTLx` register). This generates interrupts during the last frame of the block (192), allowing changes of user bits for the next block.
3. Initialize the `SPDIF_TX_CTL` register to enable the data encoding.
4. Manually set the block start bit in the data stream once per block (every 384 words). This is necessary if automatic generation of block start information is not enabled using the `SPDIF_TX_CTL.AUTO` bit = 0.

NOTE: For more information, see the "DAI Routing Capabilities" section of the *Digital Audio Interface (DAI)* chapter.

Programming the Receiver

Since the S/PDIF receiver data output is not available to the core, programming the peripheral is as simple as connecting the SRU to the on-chip (serial ports) or off-chip (DAI pins) serial devices that provide the clock and data to be decoded, and selecting the desired mode in the receiver control register. This setup can be accomplished in two steps.

1. Connect the input signal and three output signals in the SRU. The only input of the receiver is the biphase encoded stream, `SPDIFn_RX_I`. The three required output signals are the serial clock (`SPDIFn_RX_CLK`), the serial frame sync (`SPDIFn_RX_FS`), and the serial data (`SPDIFn_RX_DAT`). The high frequency clock (`SPDIFn_RX_TDMCLK`) derived from the encoded stream is also available if the system requires it.

2. Initialize the `SPDIF_RX_CTL` register to enable the data decoding. Note that this peripheral is disabled by default.

NOTE: For more information, see the "DAI Routing Capabilities" section of the *Digital Audio Interface (DAI)* chapter.

Interrupted Data Streams on the Receiver

When using the S/PDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the S/PDIF receiver's digital PLL will relock to the stream. The steps to accomplish this are described below.

1. Set up interrupts within the DAI so that the S/PDIF receiver can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the NCO. This is accomplished by setting and then clearing the `SPDIF_RX_CTL.RST` bit.
3. Return from the ISR and continue normal operation.

This method of resetting the NCO has been shown to provide extremely reliable performance when the S/PDIF inputs are interrupted or unplugged momentarily.

Debug Features

The following feature supports S/PDIF debugging.

Loopback Routing

The S/PDIF supports an internal loopback mode by using the SRU. For more information about loopback, see "Loopback Routing" in the the *Digital Audio Interface (DAI)* chapter.

ADSP-2156x SPDIF Register Descriptions

The S/PDIF module (SPDIF) contains the following registers.

Table 26-4: ADSP-2156x SPDIF Register List

Name	Description
<code>SPDIF_RX_CTL</code>	Receive Control
<code>SPDIF_RX_STAT</code>	Receive Status Register
<code>SPDIF_RX_STAT0_A</code>	Receive Status A0 Register
<code>SPDIF_RX_STAT0_B</code>	Receive Status B0 Register
<code>SPDIF_RX_STAT1_A</code>	Receive Status A1 Register
<code>SPDIF_RX_STAT1_B</code>	Receive Status B1 Register

Table 26-4: ADSP-2156x SPDIF Register List (Continued)

Name	Description
SPDIF_TX_CTL	Transmit Control Register
SPDIF_TX_STAT_A0	Transmit Status A0 Register
SPDIF_TX_STAT_A1	Transmit Status A1 Register
SPDIF_TX_STAT_A2	Transmit Status A2 Register
SPDIF_TX_STAT_A3	Transmit Status A3 Register
SPDIF_TX_STAT_A4	Transmit Status A4 Register
SPDIF_TX_STAT_A5	Transmit Status A5 Register
SPDIF_TX_STAT_B0	Transmit Status B0 Register
SPDIF_TX_STAT_B1	Transmit Status B1 Register
SPDIF_TX_STAT_B2	Transmit Status B2 Register
SPDIF_TX_STAT_B3	Transmit Status B3 Register
SPDIF_TX_STAT_B4	Transmit Status B4 Register
SPDIF_TX_STAT_B5	Transmit Status B5 Register
SPDIF_TX_UBUFF_A0	Transmit User Buffer A0 Register
SPDIF_TX_UBUFF_A1	Transmit User Buffer A1 Register
SPDIF_TX_UBUFF_A2	Transmit User Buffer A2 Register
SPDIF_TX_UBUFF_A3	Transmit User Buffer A3 Register
SPDIF_TX_UBUFF_A4	Transmit User Buffer A4 Register
SPDIF_TX_UBUFF_A5	Transmit User Buffer A5 Register
SPDIF_TX_UBUFF_B0	Transmit User Buffer B0 Register
SPDIF_TX_UBUFF_B1	Transmit User Buffer B1 Register
SPDIF_TX_UBUFF_B2	Transmit User Buffer B2 Register
SPDIF_TX_UBUFF_B3	Transmit User Buffer B3 Register
SPDIF_TX_UBUFF_B4	Transmit User Buffer B4 Register
SPDIF_TX_UBUFF_B5	Transmit User Buffer B5 Register
SPDIF_TX_USRUPDT	User Bit Update Register

Receive Control

The `SPDIF_RX_CTL` register is used to enable and control the S/PDIF receiver.

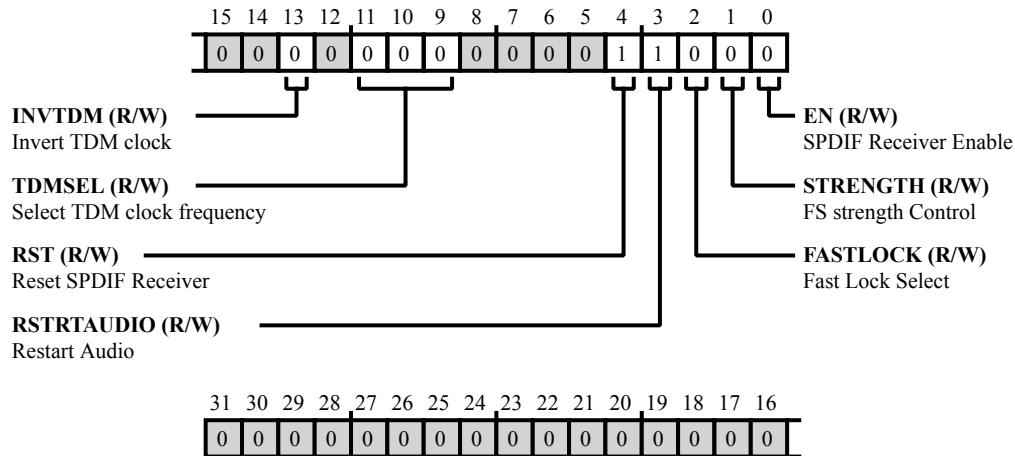


Figure 26-8: SPDIF_RX_CTL Register Diagram

Table 26-5: SPDIF_RX_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	INVTDM	Invert TDM clock. When set to 0, inverted TDM clock or non-inverted Bit clock is selected. When set to 1, non-inverted TDM clock or inverted Blit clock is selected.
11:9 (R/W)	TDMSEL	Select TDM clock frequency. Select TDM clock frequency
4 (R/W)	RST	Reset SPDIF Receiver. The <code>SPDIF_RX_CTL.RST</code> bit resets the receiver.
		0 Takes the receiver out of reset
		1 Resets the SPDIF receiver
3 (R/W)	RSTRTAUDIO	Restart Audio. The <code>SPDIF_RX_CTL.RSTRTAUDIO</code> bit restarts the audio once a re-lock has occurred. When the S/PDIF receiver loses lock the audio output is set to 0. This bit determines the behavior of the audio once lock is re-established. Audio can be manually restarted by toggling this bit high and then low.
		0 Manually restart audio
		1 Automatically restart audio

Table 26-5: SPDIF_RX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	FASTLOCK	Fast Lock Select. The <code>SPDIF_RX_CTL.FASTLOCK</code> bit allows the lock mechanism to lock at normal speed or at faster speed. This has the advantage of recovering very quickly whenever the S/PDIF receiver loses lock due to glitches in the signal. In normal mode the S/PDIF receiver locks after 64 consecutive valid samples, in fast mode the S/PDIF receiver locks after 8 consecutive valid samples
		0 Enable normal mode
		1 Enable fast mode
1 (R/W)	STRENGTH	FS strength Control. The <code>SPDIF_RX_CTL.STRENGTH</code> bit controls the strength of the bit clock and Frame sync outputs from the SPDIF receiver. In strong mode these output signals are continued (as best possible) when the receiver notices a loss-of-lock condition. Note that 'as best possible' refers to the fact that this recovered signal may not be accurate, given the loss-of-lock condition. In weak mode these output signals are interrupted as soon as the receiver notices a loss-of-lock condition.
		0 Enable strong mode
		1 Enable weak mode
0 (R/W)	EN	SPDIF Receiver Enable. When the <code>SPDIF_RX_CTL.EN</code> bit =0 the clock to SPDIF is switched off for power savings.
		0 Disable receiver
		1 Enable receiver

Receive Status Register

The `SPDIF_RX_STAT` register consists of bits that indicate the status of various functions supported by S/PDIF receiver.

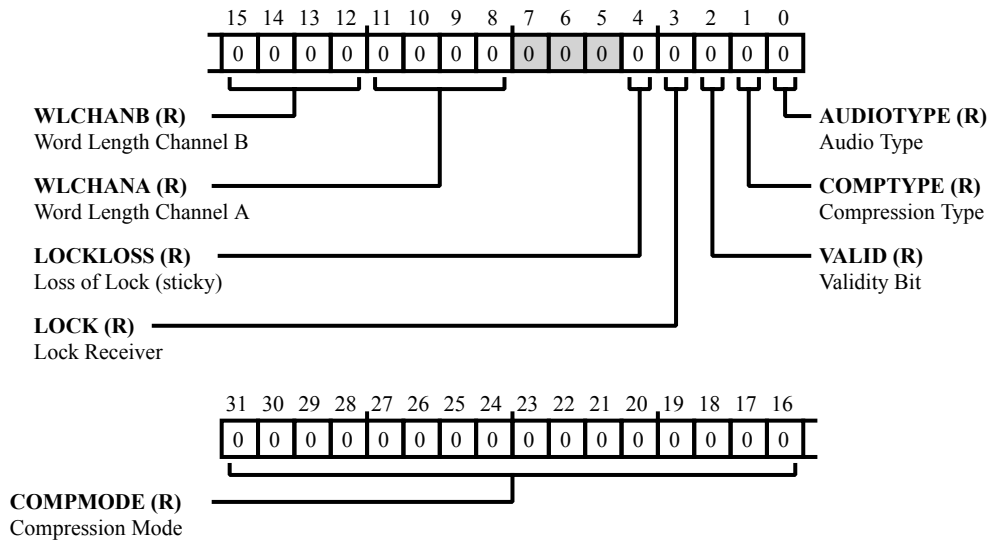


Figure 26-9: SPDIF_RX_STAT Register Diagram

Table 26-6: SPDIF_RX_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	COMPMODE	Compression Mode. The <code>SPDIF_RX_STAT.COMPMODE</code> bit field indicates the type of compression mode used in the Digital audio stream. The value in this field indicates the 16-bit burst information as specified by the IEC 62937-2 standard. Use this document to decode the value in this bit field.
15:12 (R/NW)	WLCHANB	Word Length Channel B. The <code>SPDIF_RX_STAT.WLCHANB</code> bit field indicates the S/PDIF word length for channel B. May be decoded as follows (from the S/PDIF standard).
		0 Reserved
		1 Reserved
		2 SPDIF_LENGTH_16
		3 Reserved
		4 SPDIF_LENGTH_18
		5 SPDIF_LENGTH_22
		6 Reserved

Table 26-6: SPDIF_RX_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		7 Reserved
		8 SPDIF_LENGTH_19
		9 SPDIF_LENGTH_23
		10 SPDIF_LENGTH_20
		11 SPDIF_LENGTH_24
		12 SPDIF_LENGTH_17
		13 SPDIF_LENGTH_21
		14 Reserved
		15 Reserved
11:8 (R/NW)	WLCHANA	<p>Word Length Channel A.</p> <p>The <code>SPDIF_RX_STAT.WLCHANA</code> bit indicates the S/PDIF word length for channel A. May be decoded as follows (from the S/PDIF standard).</p>
		0 Reserved
		1 Reserved
		2 SPDIF_LENGTH_16
		3 Reserved
		4 SPDIF_LENGTH_18
		5 SPDIF_LENGTH_22
		6 Reserved
		7 Reserved
		8 SPDIF_LENGTH_19
		9 SPDIF_LENGTH_23
		10 SPDIF_LENGTH_20
		11 SPDIF_LENGTH_24
		12 SPDIF_LENGTH_17
		13 SPDIF_LENGTH_21
		14 Reserved
		15 Reserved

Table 26-6: SPDIF_RX_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	LOCKLOSS	Loss of Lock (sticky). The <code>SPDIF_RX_STAT.LOCKLOSS</code> bit indicates that the system has lost lock. This is different to the lock register, as it is sticky it goes high when system loses lock, but returns to low once <code>SPDIF_RX_CTL.RSTRTAUDIO</code> bit is toggled. This is to allow the programs to poll the lock status.
		0 SPDIF receiver locked
		1 SPDIF Receiver lost lock
3 (R/NW)	LOCK	Lock Receiver. The <code>SPDIF_RX_STAT.LOCK</code> bit indicates the S/PDIF receiver has successfully locked to the S/PDIF stream and is outputting valid data.
		0 Receiver not locked
		1 Receiver locked
2 (R/NW)	VALID	Validity Bit. The <code>SPDIF_RX_STAT.VALID</code> bit indicates the ORed bits of channel A and B.
		0 Linear PCM data
		1 Non-linear audio data
1 (R/NW)	COMPTYPE	Compression Type. The <code>SPDIF_RX_STAT.COMPTYPE</code> bit indicates AC3 or DTS compression. Valid only if <code>SPDIF_RX_STAT.AUDIOTYPE</code> indicates compressed data.
		0 AC3 compressed data
		1 DTS compressed data
0 (R/NW)	AUDIOTYPE	Audio Type. The <code>SPDIF_RX_STAT.AUDIOTYPE</code> bit indicates PCM or compressed data.
		0 PCM data
		1 Compressed Data

Receive Status A0 Register

The `SPDIF_RX_STAT0_A` register holds the receive channel 0 status for bytes 0-3 for sub frame A.

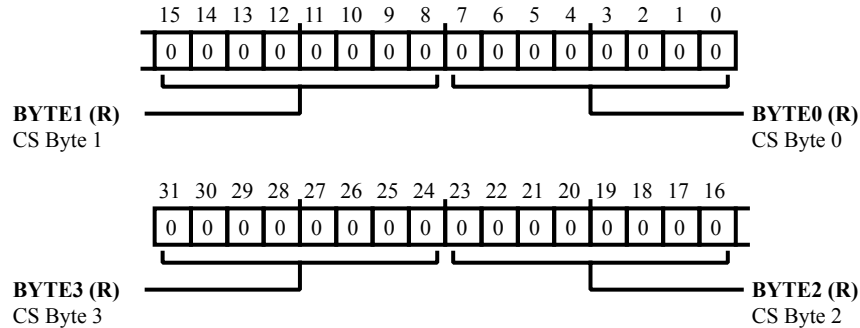


Figure 26-10: SPDIF_RX_STAT0_A Register Diagram

Table 26-7: SPDIF_RX_STAT0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	BYTE3	CS Byte 3. The <code>SPDIF_RX_STAT0_A.BYTE3</code> bit field contains byte 3 of received channel A status.
23:16 (R/NW)	BYTE2	CS Byte 2. The <code>SPDIF_RX_STAT0_A.BYTE2</code> bit field contains byte 2 of received channel A status.
15:8 (R/NW)	BYTE1	CS Byte 1. The <code>SPDIF_RX_STAT0_A.BYTE1</code> bit field contains byte 1 of received channel A status.
7:0 (R/NW)	BYTE0	CS Byte 0. The <code>SPDIF_RX_STAT0_A.BYTE0</code> bit field contains the status of byte 0-3 of received channel A.

Receive Status B0 Register

The `SPDIF_RX_STAT0_B` register holds the receive channel 0 status for bytes 0-3 for sub frame B.

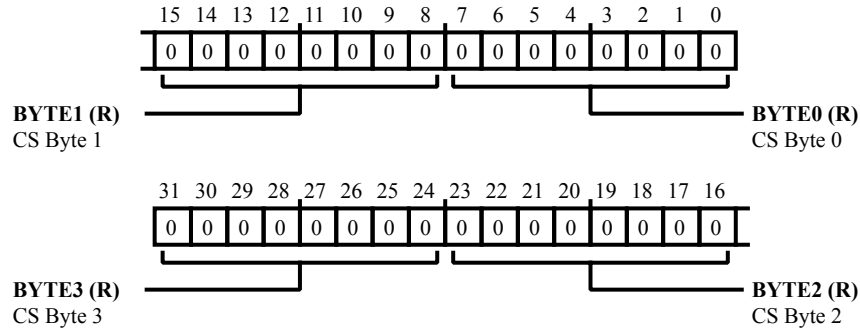


Figure 26-11: SPDIF_RX_STAT0_B Register Diagram

Table 26-8: SPDIF_RX_STAT0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	BYTE3	CS Byte 3. The <code>SPDIF_RX_STAT0_B.BYTE3</code> bit field contains byte 3 of received channel B status.
23:16 (R/NW)	BYTE2	CS Byte 2. The <code>SPDIF_RX_STAT0_B.BYTE2</code> bit field contains byte 2 of received channel B status.
15:8 (R/NW)	BYTE1	CS Byte 1. The <code>SPDIF_RX_STAT0_B.BYTE1</code> bit field contains byte 1 of received channel B status.
7:0 (R/NW)	BYTE0	CS Byte 0. The <code>SPDIF_RX_STAT0_B.BYTE0</code> bit field contains byte 0 of received channel B status.

Receive Status A1 Register

The `SPDIF_RX_STAT1_A` register holds the receive channel 1 status for byte 4 for sub frame A.

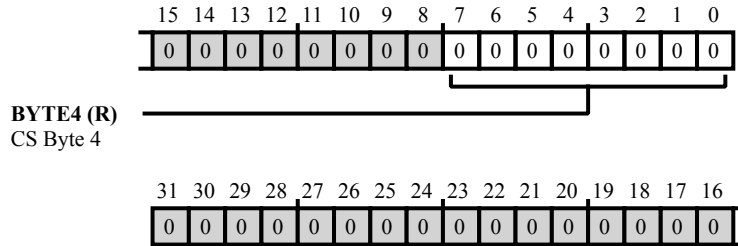


Figure 26-12: SPDIF_RX_STAT1_A Register Diagram

Table 26-9: SPDIF_RX_STAT1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	BYTE4	CS Byte 4. The <code>SPDIF_RX_STAT1_A.BYTE4</code> bit field contains byte 4 of received channel A status.

Receive Status B1 Register

The `SPDIF_RX_STAT1_B` register holds the receive channel 1 status for byte 4 for sub frame B.

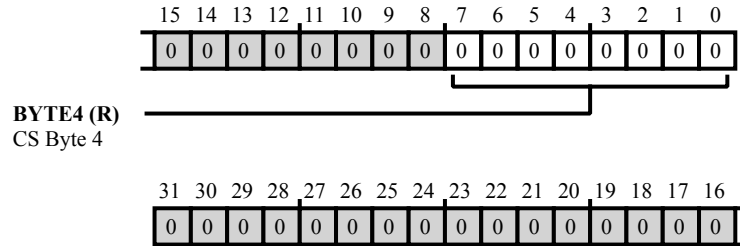


Figure 26-13: SPDIF_RX_STAT1_B Register Diagram

Table 26-10: SPDIF_RX_STAT1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	BYTE4	CS Byte 4. The <code>SPDIF_RX_STAT1_B.BYTE4</code> bit field contains byte 4 of received channel B status.

Transmit Control Register

The `SPDIF_TX_CTL` register provides bits to enable/disable the transmitter and configure several options related to data transmission.

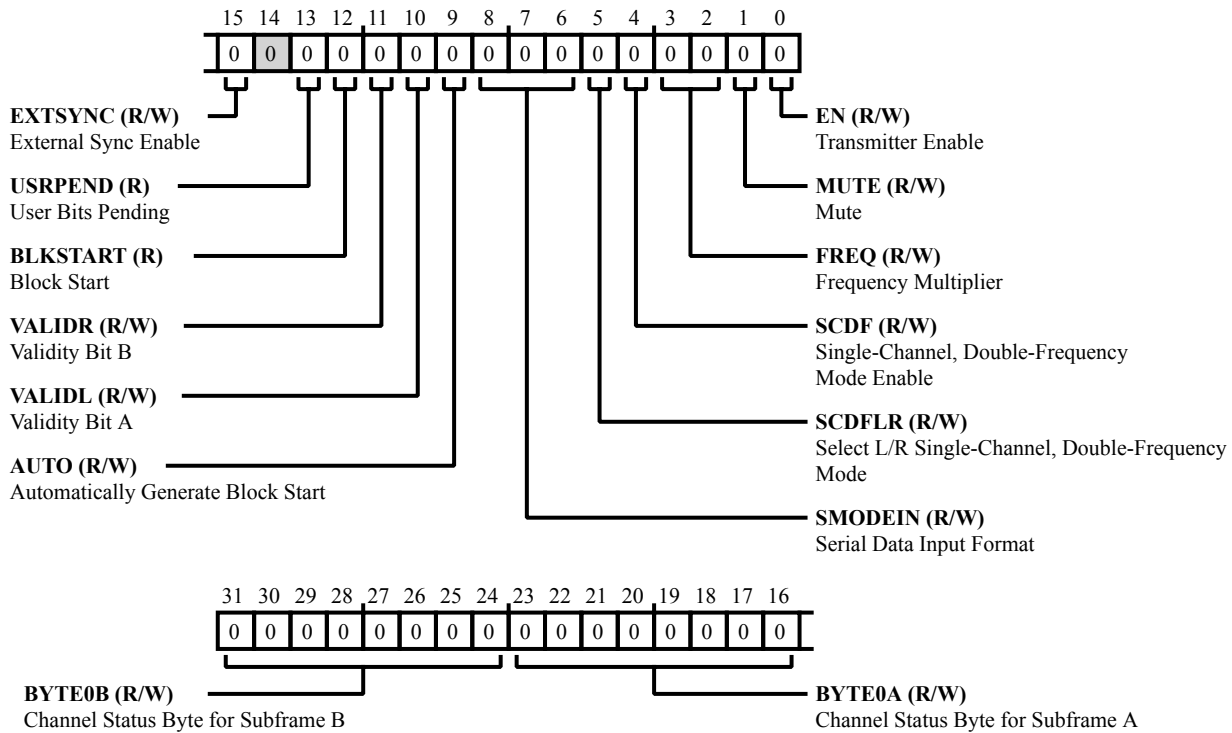


Figure 26-14: SPDIF_TX_CTL Register Diagram

Table 26-11: SPDIF_TX_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE0B	Channel Status Byte for Subframe B. The <code>SPDIF_TX_CTL.BYTE0B</code> bit field contains the channel status for the second bytes 95.
23:16 (R/W)	BYTE0A	Channel Status Byte for Subframe A. The <code>SPDIF_TX_CTL.BYTE0A</code> bit field contains the channel status for the first bytes 40.
15 (R/W)	EXTSYNC	External Sync Enable. When the <code>SPDIF_TX_CTL.EXTSYNC</code> bit is set (regardless of the <code>SPDIF_TX_CTL.AUTO</code> bit setting) the internal frame counter is set to zero at an internal LRCLK rising edge followed by an <code>EXTSYNC_I</code> rising edge.

Table 26-11: SPDIF_TX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/NW)	USRPEND	User Bits Pending. The SPDIF_TX_CTL.USRPEND bit is set if the update of the internal buffer from the Transmit User Bits Buffer registers has not yet completed.
12 (R/NW)	BLKSTART	Block Start. The SPDIF_TX_CTL.BLKSTART bit is a status bit that indicates block start (when the SPDIF_TX_CTL.AUTO bit = 1).
		0 Current word is not block start
		1 Current word is block start
11 (R/W)	VALIDR	Validity Bit B. Use the SPDIF_TX_CTL.VALIDR bit with the channel status buffer.
10 (R/W)	VALIDL	Validity Bit A. The SPDIF_TX_CTL.VALIDL bit either manually start block transfer according to input stream status bits or automatically start block transfer. Use the SPDIF_TX_CTL.VALIDL bit with the channel status buffer.
		0 Manually start block transfer
		1 Automatically start block transfer
9 (R/W)	AUTO	Automatically Generate Block Start. When enabled, the transmitter is in standalone mode where it inserts block start, channel status, and validity bits on its own. If the channel status or validity buffer needs to be enabled (after the DAI programming is complete), first write to the buffers with the required data and then enable the buffers by setting the SPDIF_TX_CTL.AUTO bit.
8:6 (R/W)	SMODEIN	Serial Data Input Format. The SPDIF_TX_CTL.SMODEIN bit selects the data input format.
		0 Left-justified
		1 I2S
		2 Reserved
		3 Reserved
		4 Right-justified, 24 bits
		5 Right-justified, 20 bits
		6 Right-justified, 18 bits
7 Right-justified, 16 bits		

Table 26-11: SPDIF_TX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SCDFLR	Select L/R Single-Channel, Double-Frequency Mode. The <code>SPDIF_TX_CTL.SCDFLR</code> bit selects the left or right channel in SCDF mode.
		0 Left channel
		1 Right channel
4 (R/W)	SCDF	Single-Channel, Double-Frequency Mode Enable. The <code>SPDIF_TX_CTL.SCDF</code> bit enables single-channel, double-frequency mode.
		0 Two-channel mode
		1 SCDF mode
3:2 (R/W)	FREQ	Frequency Multiplier. The <code>SPDIF_TX_CTL.FREQ</code> bit field sets the oversampling ratio.
		0 256 x frame sync oversampling
		1 384 x frame sync oversampling
		2-3 Reserved
1 (R/W)	MUTE	Mute. The <code>SPDIF_TX_CTL.MUTE</code> bit mutes the serial data output.
		0 Disable Mute
		1 Enable Mute
0 (R/W)	EN	Transmitter Enable. The <code>SPDIF_TX_CTL.EN</code> bit enables the transmitter and resets the control registers to their defaults.
		0 Transmitter disabled
		1 Transmitter enabled

Transmit Status A0 Register

The `SPDIF_TX_STAT_A0` register holds the transmit channel 0 status for bytes 1-4 for sub frame A.

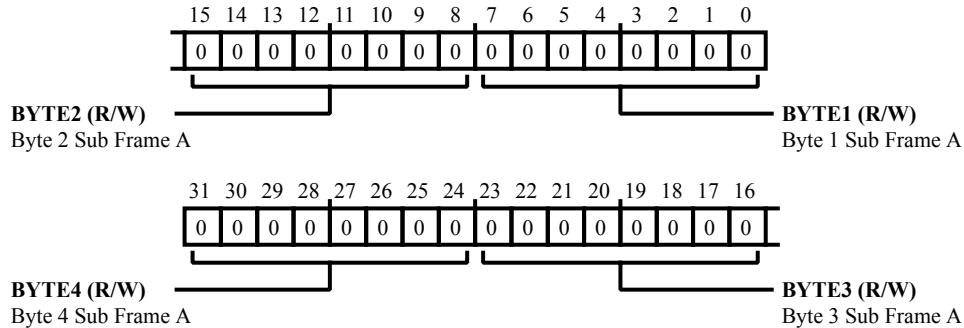


Figure 26-15: `SPDIF_TX_STAT_A0` Register Diagram

Table 26-12: `SPDIF_TX_STAT_A0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE4	Byte 4 Sub Frame A. The <code>SPDIF_TX_STAT_A0.BYTE4</code> bit field holds the transmit channel 0 status for byte 4 for sub frame A.
23:16 (R/W)	BYTE3	Byte 3 Sub Frame A. The <code>SPDIF_TX_STAT_A0.BYTE3</code> bit field holds the transmit channel 0 status for byte 3 for sub frame A.
15:8 (R/W)	BYTE2	Byte 2 Sub Frame A. The <code>SPDIF_TX_STAT_A0.BYTE2</code> bit field holds the transmit channel 0 status for byte 2 for sub frame A.
7:0 (R/W)	BYTE1	Byte 1 Sub Frame A. The <code>SPDIF_TX_STAT_A0.BYTE1</code> bit field holds the transmit channel 0 status for byte 1 for sub frame A.

Transmit Status A1 Register

The `SPDIF_TX_STAT_A1` register holds the transmit status for bytes 5-8 for sub frame A.

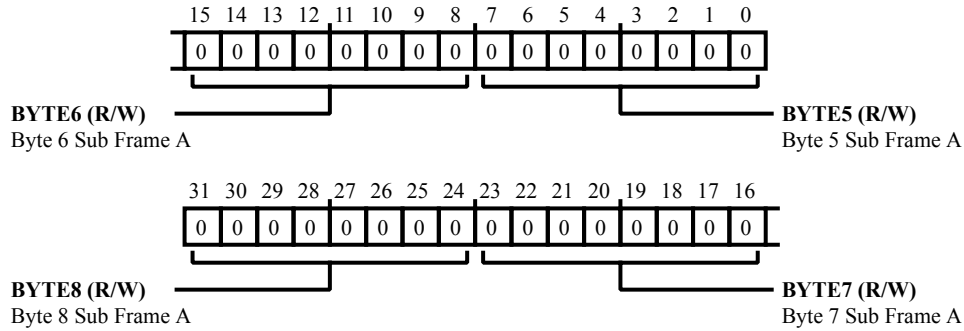


Figure 26-16: `SPDIF_TX_STAT_A1` Register Diagram

Table 26-13: `SPDIF_TX_STAT_A1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE8	Byte 8 Sub Frame A. The <code>SPDIF_TX_STAT_A1</code> . <code>BYTE8</code> bit field contains transmit status for byte 8 of sub frame A.
23:16 (R/W)	BYTE7	Byte 7 Sub Frame A. The <code>SPDIF_TX_STAT_A1</code> . <code>BYTE7</code> bit field contains transmit status for byte 7 of sub frame A.
15:8 (R/W)	BYTE6	Byte 6 Sub Frame A. The <code>SPDIF_TX_STAT_A1</code> . <code>BYTE6</code> bit field contains transmit status for byte 6 of sub frame A.
7:0 (R/W)	BYTE5	Byte 5 Sub Frame A. The <code>SPDIF_TX_STAT_A1</code> . <code>BYTE5</code> bit field contains transmit status for byte 5 of sub frame A.

Transmit Status A2 Register

The `SPDIF_TX_STAT_A2` register holds the transmit status for bytes 9-12 for sub frame A.

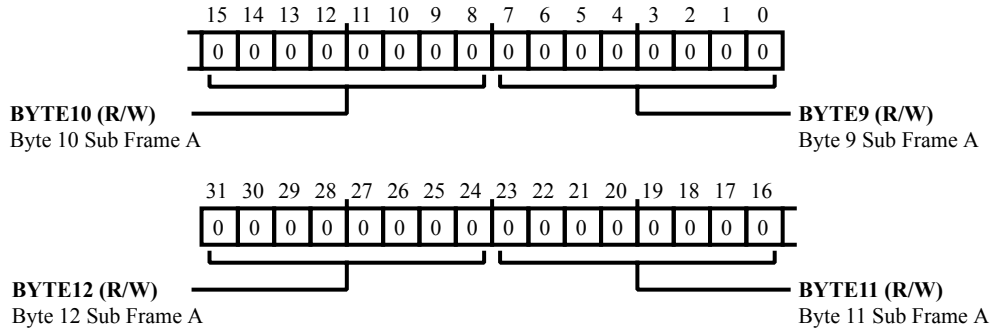


Figure 26-17: `SPDIF_TX_STAT_A2` Register Diagram

Table 26-14: `SPDIF_TX_STAT_A2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE12	Byte 12 Sub Frame A. The <code>SPDIF_TX_STAT_A2.BYTE12</code> bit field contains transmit status for byte 12 of sub frame A.
23:16 (R/W)	BYTE11	Byte 11 Sub Frame A. The <code>SPDIF_TX_STAT_A2.BYTE11</code> bit field contains transmit status for byte 11 of sub frame A.
15:8 (R/W)	BYTE10	Byte 10 Sub Frame A. The <code>SPDIF_TX_STAT_A2.BYTE10</code> bit field contains transmit status for byte 10 of sub frame A.
7:0 (R/W)	BYTE9	Byte 9 Sub Frame A. The <code>SPDIF_TX_STAT_A2.BYTE9</code> bit field contains transmit status for byte 9 of sub frame A.

Transmit Status A3 Register

The `SPDIF_TX_STAT_A3` register holds the transmit status for bytes 13-16 for sub frame A.

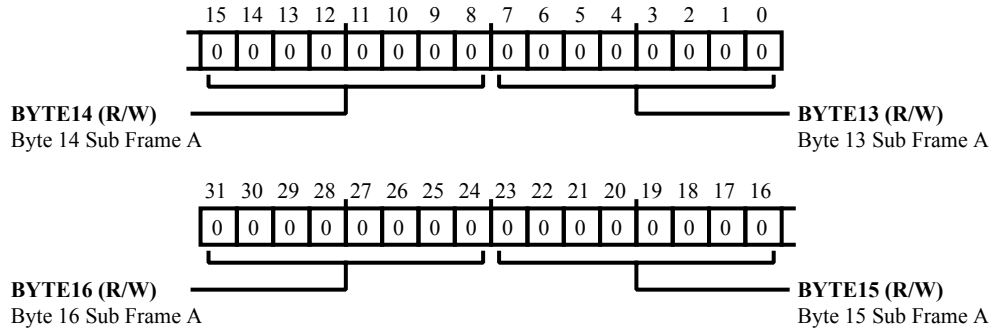


Figure 26-18: `SPDIF_TX_STAT_A3` Register Diagram

Table 26-15: `SPDIF_TX_STAT_A3` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE16	Byte 16 Sub Frame A. The <code>SPDIF_TX_STAT_A3.BYTE16</code> bit field contains transmit status for byte 16 of sub frame A.
23:16 (R/W)	BYTE15	Byte 15 Sub Frame A. The <code>SPDIF_TX_STAT_A3.BYTE15</code> bit field contains transmit status for byte 15 of sub frame A.
15:8 (R/W)	BYTE14	Byte 14 Sub Frame A. The <code>SPDIF_TX_STAT_A3.BYTE14</code> bit field contains transmit status for byte 14 of sub frame A.
7:0 (R/W)	BYTE13	Byte 13 Sub Frame A. The <code>SPDIF_TX_STAT_A3.BYTE13</code> bit field contains transmit status for byte 13 of sub frame A.

Transmit Status A4 Register

The `SPDIF_TX_STAT_A4` register holds the transmit status for bytes 17-20 for sub frame A.

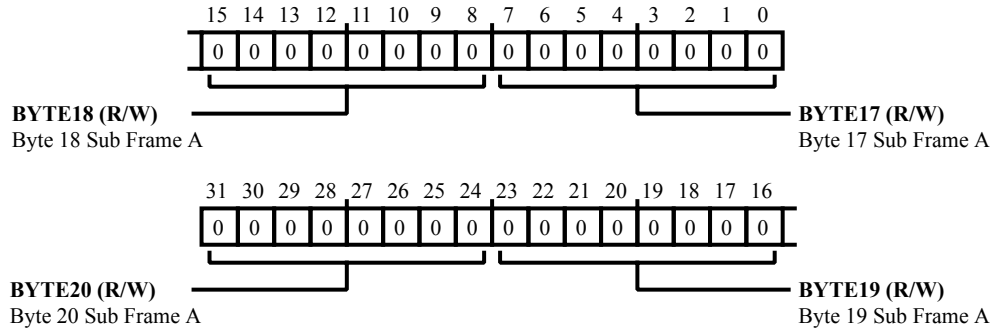


Figure 26-19: `SPDIF_TX_STAT_A4` Register Diagram

Table 26-16: `SPDIF_TX_STAT_A4` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE20	Byte 20 Sub Frame A. The <code>SPDIF_TX_STAT_A4</code> . <code>BYTE20</code> bit field contains transmit status for byte 20 of sub frame A.
23:16 (R/W)	BYTE19	Byte 19 Sub Frame A. The <code>SPDIF_TX_STAT_A4</code> . <code>BYTE19</code> bit field contains transmit status for byte 19 of sub frame A.
15:8 (R/W)	BYTE18	Byte 18 Sub Frame A. The <code>SPDIF_TX_STAT_A4</code> . <code>BYTE18</code> bit field contains transmit status for byte 18 of sub frame A.
7:0 (R/W)	BYTE17	Byte 17 Sub Frame A. The <code>SPDIF_TX_STAT_A4</code> . <code>BYTE17</code> bit field contains transmit status for byte 17 of sub frame A.

Transmit Status A5 Register

The `SPDIF_TX_STAT_A5` register holds the transmit status for bytes 21-23 for sub frame A.

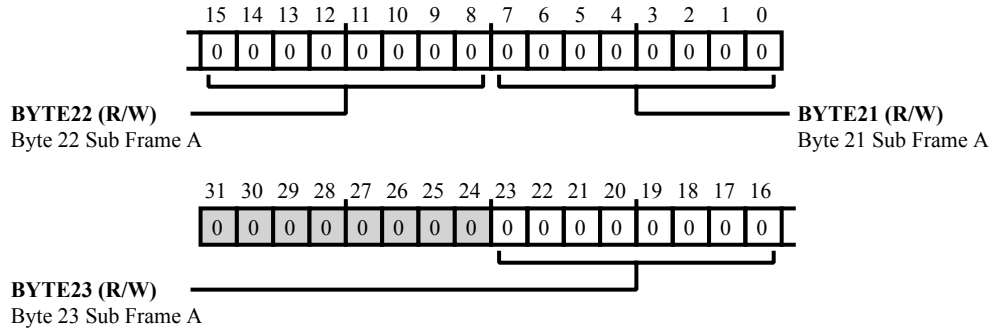


Figure 26-20: SPDIF_TX_STAT_A5 Register Diagram

Table 26-17: SPDIF_TX_STAT_A5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	BYTE23	Byte 23 Sub Frame A. The <code>SPDIF_TX_STAT_A5.BYTE23</code> bit field contains transmit status for byte 23 of sub frame A.
15:8 (R/W)	BYTE22	Byte 22 Sub Frame A. The <code>SPDIF_TX_STAT_A5.BYTE22</code> bit field contains transmit status for byte 22 of sub frame A.
7:0 (R/W)	BYTE21	Byte 21 Sub Frame A. The <code>SPDIF_TX_STAT_A5.BYTE21</code> bit field contains transmit status for byte 21 of sub frame A.

Transmit Status B0 Register

The `SPDIF_TX_STAT_B0` register holds the transmit channel 0 status for bytes 1-4 for sub frame B.

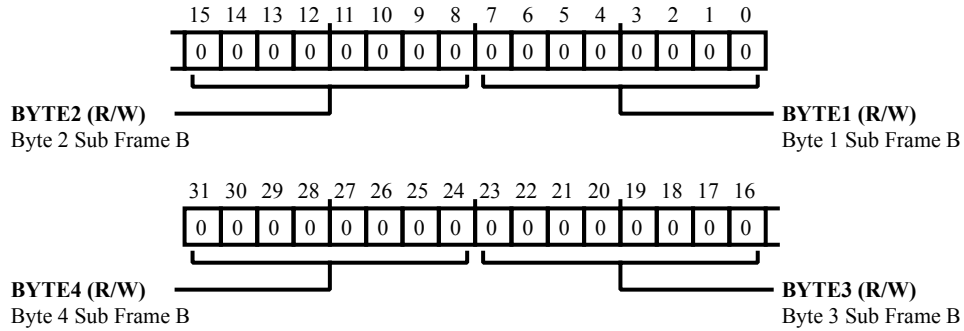


Figure 26-21: SPDIF_TX_STAT_B0 Register Diagram

Table 26-18: SPDIF_TX_STAT_B0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE4	Byte 4 Sub Frame B. The <code>SPDIF_TX_STAT_B0.BYTE4</code> bit field holds the transmit channel 0 status for byte 4 for sub frame B.
23:16 (R/W)	BYTE3	Byte 3 Sub Frame B. The <code>SPDIF_TX_STAT_B0.BYTE3</code> bit field holds the transmit channel 0 status for byte 3 for sub frame B.
15:8 (R/W)	BYTE2	Byte 2 Sub Frame B. The <code>SPDIF_TX_STAT_B0.BYTE2</code> bit field holds the transmit channel 0 status for byte 2 for sub frame B.
7:0 (R/W)	BYTE1	Byte 1 Sub Frame B. The <code>SPDIF_TX_STAT_B0.BYTE1</code> bit field holds the transmit channel 0 status for byte 1 for sub frame B.

Transmit Status B1 Register

The `SPDIF_TX_STAT_B1` register holds the transmit status for bytes 5-8 for sub frame B.

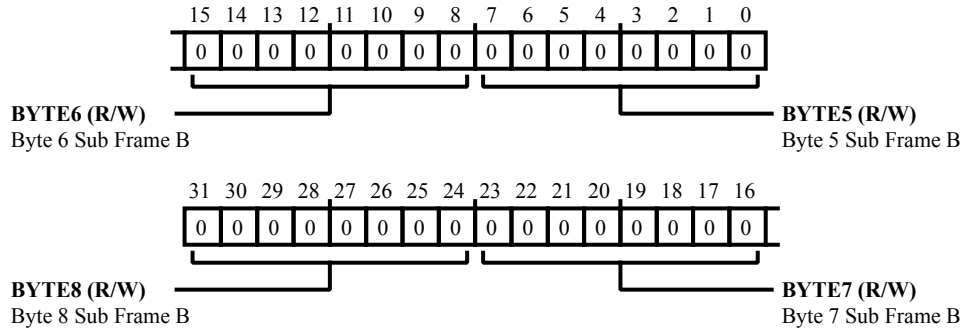


Figure 26-22: `SPDIF_TX_STAT_B1` Register Diagram

Table 26-19: `SPDIF_TX_STAT_B1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE8	Byte 8 Sub Frame B. The <code>SPDIF_TX_STAT_B1</code> . <code>BYTE8</code> bit field contains transmit status for byte 8 of sub frame B.
23:16 (R/W)	BYTE7	Byte 7 Sub Frame B. The <code>SPDIF_TX_STAT_B1</code> . <code>BYTE7</code> bit field contains transmit status for byte 7 of sub frame B.
15:8 (R/W)	BYTE6	Byte 6 Sub Frame B. The <code>SPDIF_TX_STAT_B1</code> . <code>BYTE6</code> bit field contains transmit status for byte 6 of sub frame B.
7:0 (R/W)	BYTE5	Byte 5 Sub Frame B. The <code>SPDIF_TX_STAT_B1</code> . <code>BYTE5</code> bit field contains transmit status for byte 5 of sub frame B.

Transmit Status B2 Register

The `SPDIF_TX_STAT_B2` register holds the transmit status for bytes 9-12 for sub frame B.

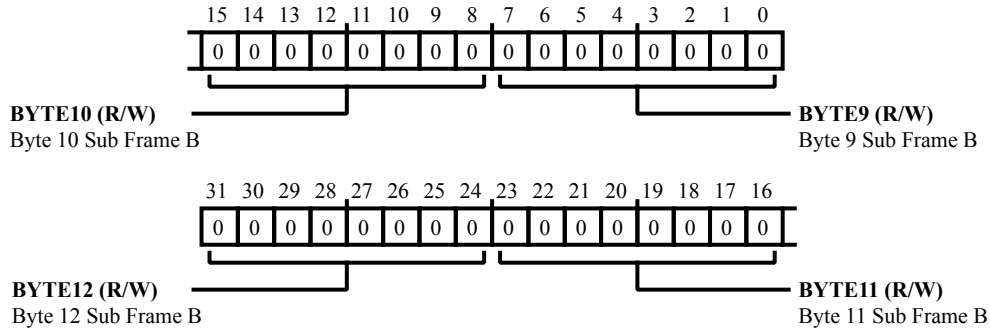


Figure 26-23: `SPDIF_TX_STAT_B2` Register Diagram

Table 26-20: `SPDIF_TX_STAT_B2` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE12	Byte 12 Sub Frame B. The <code>SPDIF_TX_STAT_B2.BYTE12</code> bit field contains transmit status for byte 12 of sub frame B.
23:16 (R/W)	BYTE11	Byte 11 Sub Frame B. The <code>SPDIF_TX_STAT_B2.BYTE11</code> bit field contains transmit status for byte 11 of sub frame B.
15:8 (R/W)	BYTE10	Byte 10 Sub Frame B. The <code>SPDIF_TX_STAT_B2.BYTE10</code> bit field contains transmit status for byte 10 of sub frame B.
7:0 (R/W)	BYTE9	Byte 9 Sub Frame B. The <code>SPDIF_TX_STAT_B2.BYTE9</code> bit field contains transmit status for byte 9 of sub frame B.

Transmit Status B3 Register

The `SPDIF_TX_STAT_B3` register holds the transmit status for bytes 13-16 for sub frame B.

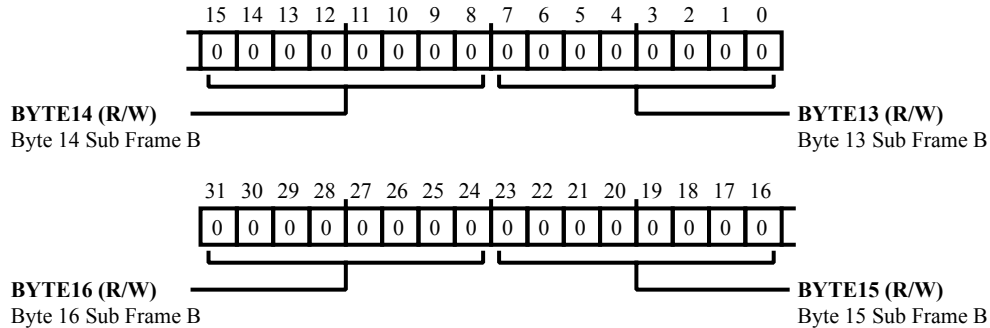


Figure 26-24: `SPDIF_TX_STAT_B3` Register Diagram

Table 26-21: `SPDIF_TX_STAT_B3` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE16	Byte 16 Sub Frame B. The <code>SPDIF_TX_STAT_B3.BYTE16</code> bit field contains transmit status for byte 16 of sub frame B.
23:16 (R/W)	BYTE15	Byte 15 Sub Frame B. The <code>SPDIF_TX_STAT_B3.BYTE15</code> bit field contains transmit status for byte 15 of sub frame B.
15:8 (R/W)	BYTE14	Byte 14 Sub Frame B. The <code>SPDIF_TX_STAT_B3.BYTE14</code> bit field contains transmit status for byte 14 of sub frame B.
7:0 (R/W)	BYTE13	Byte 13 Sub Frame B. The <code>SPDIF_TX_STAT_B3.BYTE13</code> bit field contains transmit status for byte 13 of sub frame B.

Transmit Status B4 Register

The `SPDIF_TX_STAT_B4` register holds the transmit status for bytes 17-20 for sub frame B.

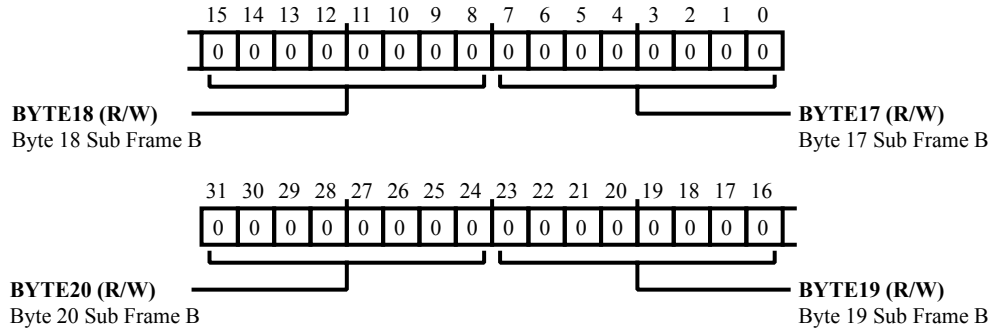


Figure 26-25: `SPDIF_TX_STAT_B4` Register Diagram

Table 26-22: `SPDIF_TX_STAT_B4` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE20	Byte 20 Sub Frame B. The <code>SPDIF_TX_STAT_B4.BYTE20</code> bit field contains transmit status for byte 20 of sub frame B.
23:16 (R/W)	BYTE19	Byte 19 Sub Frame B. The <code>SPDIF_TX_STAT_B4.BYTE19</code> bit field contains transmit status for byte 19 of sub frame B.
15:8 (R/W)	BYTE18	Byte 18 Sub Frame B. The <code>SPDIF_TX_STAT_B4.BYTE18</code> bit field contains transmit status for byte 18 of sub frame B.
7:0 (R/W)	BYTE17	Byte 17 Sub Frame B. The <code>SPDIF_TX_STAT_B4.BYTE17</code> bit field contains transmit status for byte 17 of sub frame B.

Transmit Status B5 Register

The `SPDIF_TX_STAT_B5` register holds the transmit status for bytes 21-23 for sub frame B.

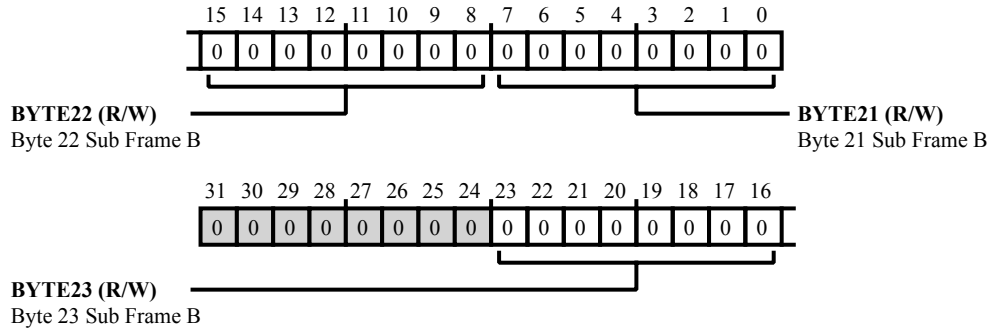


Figure 26-26: `SPDIF_TX_STAT_B5` Register Diagram

Table 26-23: `SPDIF_TX_STAT_B5` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	BYTE23	Byte 23 Sub Frame B. The <code>SPDIF_TX_STAT_B5.BYTE23</code> bit field contains transmit status for byte 23 of sub frame B.
15:8 (R/W)	BYTE22	Byte 22 Sub Frame B. The <code>SPDIF_TX_STAT_B5.BYTE22</code> bit field contains transmit status for byte 22 of sub frame B.
7:0 (R/W)	BYTE21	Byte 21 Sub Frame B. The <code>SPDIF_TX_STAT_B5.BYTE21</code> bit field contains transmit status for byte 21 of sub frame B.

Transmit User Buffer A0 Register

The `SPDIF_TX_UBUFF_A0` register holds the transmit user buffer data for bytes 0-3 for sub frame A.

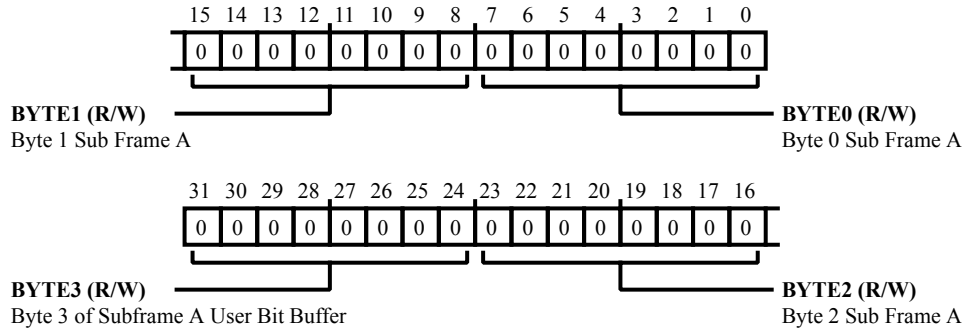


Figure 26-27: SPDIF_TX_UBUFF_A0 Register Diagram

Table 26-24: SPDIF_TX_UBUFF_A0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE3	Byte 3 of Subframe A User Bit Buffer. The <code>SPDIF_TX_UBUFF_A0.BYTE3</code> bit field contains user bit data for byte 0 of sub frame A.
23:16 (R/W)	BYTE2	Byte 2 Sub Frame A. The <code>SPDIF_TX_UBUFF_A0.BYTE2</code> bit field contains user bit data for byte 2 of sub frame A.
15:8 (R/W)	BYTE1	Byte 1 Sub Frame A. The <code>SPDIF_TX_UBUFF_A0.BYTE1</code> bit field contains user bit data for byte 1 of sub frame A.
7:0 (R/W)	BYTE0	Byte 0 Sub Frame A. The <code>SPDIF_TX_UBUFF_A0.BYTE0</code> bit field contains user bit data for byte 0 of sub frame A.

Transmit User Buffer A1 Register

The `SPDIF_TX_UBUFF_A1` register holds the transmit user buffer data for bytes 4-7 for sub frame A.

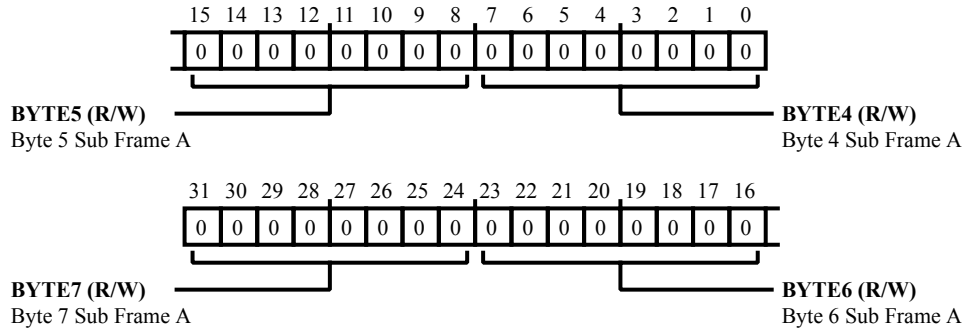


Figure 26-28: SPDIF_TX_UBUFF_A1 Register Diagram

Table 26-25: SPDIF_TX_UBUFF_A1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE7	Byte 7 Sub Frame A. The <code>SPDIF_TX_UBUFF_A1</code> . <code>BYTE7</code> bit field contains user bit data for byte 7 of sub frame A.
23:16 (R/W)	BYTE6	Byte 6 Sub Frame A. The <code>SPDIF_TX_UBUFF_A1</code> . <code>BYTE6</code> bit field contains user bit data for byte 6 of sub frame A.
15:8 (R/W)	BYTE5	Byte 5 Sub Frame A. The <code>SPDIF_TX_UBUFF_A1</code> . <code>BYTE5</code> bit field contains user bit data for byte 5 of sub frame A.
7:0 (R/W)	BYTE4	Byte 4 Sub Frame A. The <code>SPDIF_TX_UBUFF_A1</code> . <code>BYTE4</code> bit field contains user bit data for byte 0 of sub frame A.

Transmit User Buffer A2 Register

The `SPDIF_TX_UBUFF_A2` register holds the transmit user buffer data for bytes 8-11 for sub frame A.

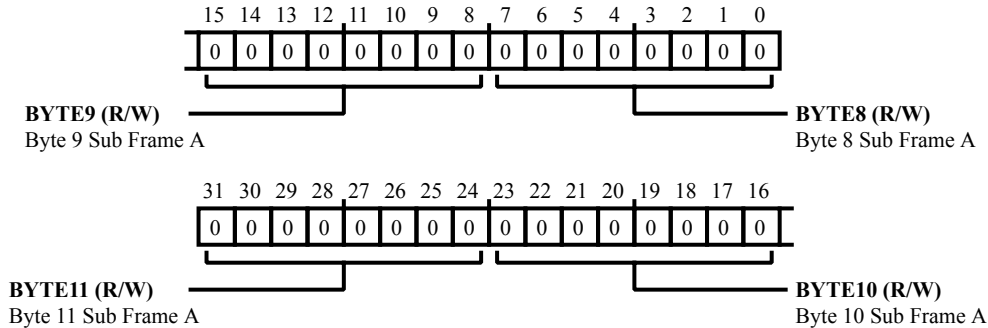


Figure 26-29: SPDIF_TX_UBUFF_A2 Register Diagram

Table 26-26: SPDIF_TX_UBUFF_A2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE11	Byte 11 Sub Frame A. The <code>SPDIF_TX_UBUFF_A2.BYTE11</code> bit field contains user bit data for byte 11 of sub frame A.
23:16 (R/W)	BYTE10	Byte 10 Sub Frame A. The <code>SPDIF_TX_UBUFF_A2.BYTE10</code> bit field contains user bit data for byte 10 of sub frame A.
15:8 (R/W)	BYTE9	Byte 9 Sub Frame A. The <code>SPDIF_TX_UBUFF_A2.BYTE9</code> bit field contains user bit data for byte 9 of sub frame A.
7:0 (R/W)	BYTE8	Byte 8 Sub Frame A. The <code>SPDIF_TX_UBUFF_A2.BYTE8</code> bit field contains user bit data for byte 8 of sub frame A.

Transmit User Buffer A3 Register

The `SPDIF_TX_UBUFF_A3` register holds the transmit user buffer data for bytes 12-15 for sub frame A.

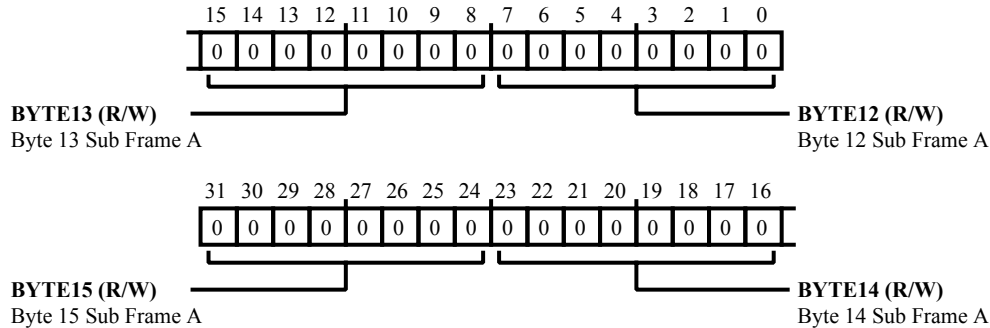


Figure 26-30: SPDIF_TX_UBUFF_A3 Register Diagram

Table 26-27: SPDIF_TX_UBUFF_A3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE15	Byte 15 Sub Frame A. The <code>SPDIF_TX_UBUFF_A3.BYTE15</code> bit field contains user bit data for byte 15 of sub frame A.
23:16 (R/W)	BYTE14	Byte 14 Sub Frame A. The <code>SPDIF_TX_UBUFF_A3.BYTE14</code> bit field contains user bit data for byte 14 of sub frame A.
15:8 (R/W)	BYTE13	Byte 13 Sub Frame A. The <code>SPDIF_TX_UBUFF_A3.BYTE13</code> bit field contains user bit data for byte 13 of sub frame A.
7:0 (R/W)	BYTE12	Byte 12 Sub Frame A. The <code>SPDIF_TX_UBUFF_A3.BYTE12</code> bit field contains user bit data for byte 12 of sub frame A.

Transmit User Buffer A4 Register

The `SPDIF_TX_UBUFF_A4` register holds the transmit user buffer data for bytes 16-19 for sub frame A.

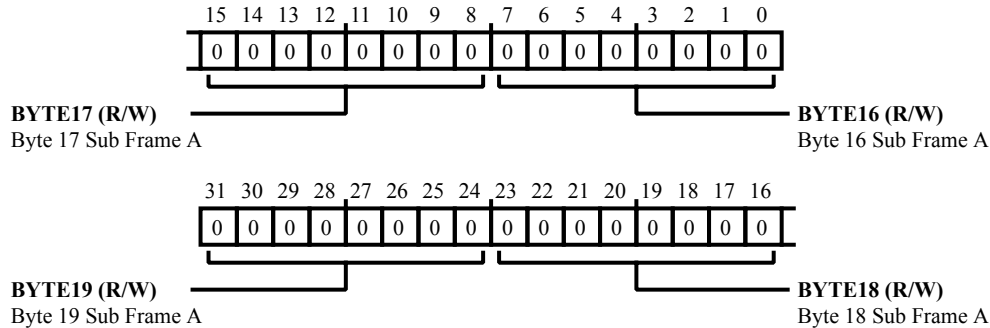


Figure 26-31: SPDIF_TX_UBUFF_A4 Register Diagram

Table 26-28: SPDIF_TX_UBUFF_A4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE19	Byte 19 Sub Frame A. The <code>SPDIF_TX_UBUFF_A4.BYTE19</code> bit field contains user bit data for byte 19 of sub frame A.
23:16 (R/W)	BYTE18	Byte 18 Sub Frame A. The <code>SPDIF_TX_UBUFF_A4.BYTE18</code> bit field contains user bit data for byte 18 of sub frame A.
15:8 (R/W)	BYTE17	Byte 17 Sub Frame A. The <code>SPDIF_TX_UBUFF_A4.BYTE17</code> bit field contains user bit data for byte 17 of sub frame A.
7:0 (R/W)	BYTE16	Byte 16 Sub Frame A. The <code>SPDIF_TX_UBUFF_A4.BYTE16</code> bit field contains user bit data for byte 16 of sub frame A.

Transmit User Buffer A5 Register

The `SPDIF_TX_UBUFF_A5` register holds the transmit user buffer data for bytes 20-23 for sub frame A.

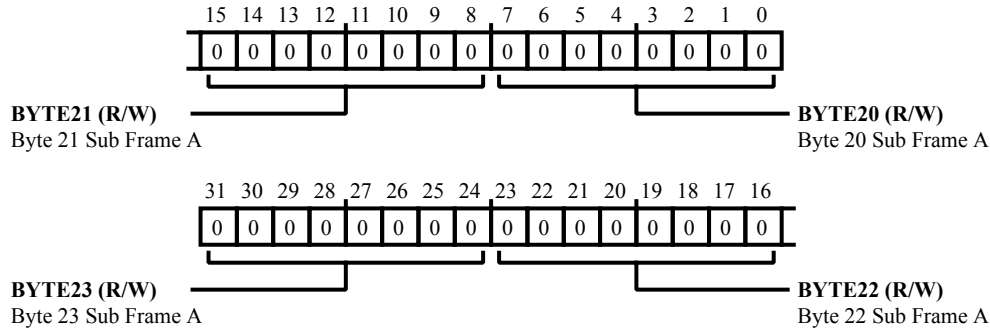


Figure 26-32: SPDIF_TX_UBUFF_A5 Register Diagram

Table 26-29: SPDIF_TX_UBUFF_A5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE23	Byte 23 Sub Frame A. The <code>SPDIF_TX_UBUFF_A5.BYTE23</code> bit field contains user bit data for byte 23 of sub frame A.
23:16 (R/W)	BYTE22	Byte 22 Sub Frame A. The <code>SPDIF_TX_UBUFF_A5.BYTE22</code> bit field contains user bit data for byte 22 of sub frame A.
15:8 (R/W)	BYTE21	Byte 21 Sub Frame A. The <code>SPDIF_TX_UBUFF_A5.BYTE21</code> bit field contains user bit data for byte 21 of sub frame A.
7:0 (R/W)	BYTE20	Byte 20 Sub Frame A. The <code>SPDIF_TX_UBUFF_A5.BYTE20</code> bit field contains user bit data for byte 20 of sub frame A.

Transmit User Buffer B0 Register

The `SPDIF_TX_UBUFF_B0` register holds the transmit user buffer data for bytes 0-3 for sub frame B.

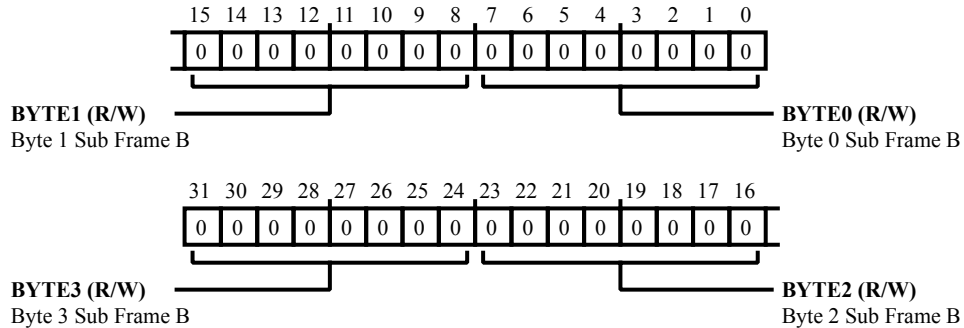


Figure 26-33: SPDIF_TX_UBUFF_B0 Register Diagram

Table 26-30: SPDIF_TX_UBUFF_B0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE3	Byte 3 Sub Frame B. The <code>SPDIF_TX_UBUFF_B0.BYTE3</code> bit field contains user bit data for byte 3 of sub frame B.
23:16 (R/W)	BYTE2	Byte 2 Sub Frame B. The <code>SPDIF_TX_UBUFF_B0.BYTE2</code> bit field contains user bit data for byte 2 of sub frame B.
15:8 (R/W)	BYTE1	Byte 1 Sub Frame B. The <code>SPDIF_TX_UBUFF_B0.BYTE1</code> bit field contains user bit data for byte 1 of sub frame B.
7:0 (R/W)	BYTE0	Byte 0 Sub Frame B. The <code>SPDIF_TX_UBUFF_B0.BYTE0</code> bit field contains user bit data for byte 0 of sub frame B.

Transmit User Buffer B1 Register

The `SPDIF_TX_UBUFF_B1` register holds the transmit user buffer data for bytes 4-7 for sub frame B.

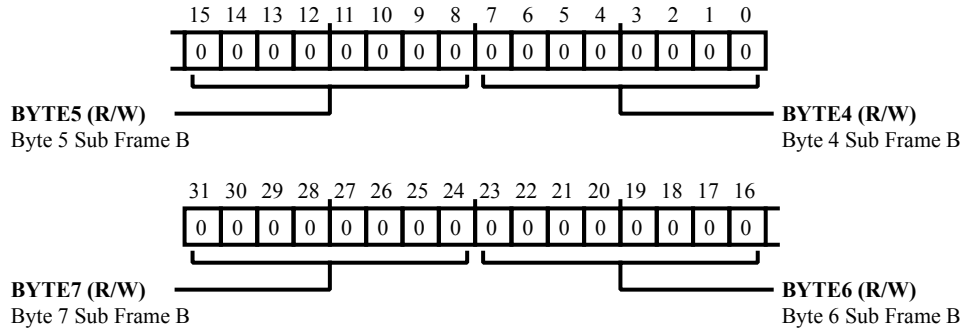


Figure 26-34: SPDIF_TX_UBUFF_B1 Register Diagram

Table 26-31: SPDIF_TX_UBUFF_B1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE7	Byte 7 Sub Frame B. The <code>SPDIF_TX_UBUFF_B1.BYTE7</code> bit field contains user bit data for byte 7 of sub frame B.
23:16 (R/W)	BYTE6	Byte 6 Sub Frame B. The <code>SPDIF_TX_UBUFF_B1.BYTE6</code> bit field contains user bit data for byte 6 of sub frame B.
15:8 (R/W)	BYTE5	Byte 5 Sub Frame B. The <code>SPDIF_TX_UBUFF_B1.BYTE5</code> bit field contains user bit data for byte 5 of sub frame B.
7:0 (R/W)	BYTE4	Byte 4 Sub Frame B. The <code>SPDIF_TX_UBUFF_B1.BYTE4</code> bit field contains user bit data for byte 4 of sub frame B.

Transmit User Buffer B2 Register

The `SPDIF_TX_UBUFF_B2` register holds the transmit user buffer data for bytes 8-11 for sub frame B.

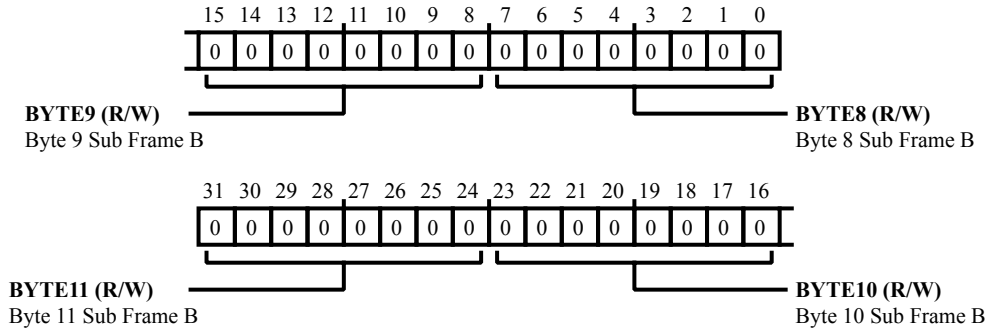


Figure 26-35: SPDIF_TX_UBUFF_B2 Register Diagram

Table 26-32: SPDIF_TX_UBUFF_B2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE11	Byte 11 Sub Frame B. The <code>SPDIF_TX_UBUFF_B2.BYTE11</code> bit field contains user bit data for byte 11 of sub frame B.
23:16 (R/W)	BYTE10	Byte 10 Sub Frame B. The <code>SPDIF_TX_UBUFF_B2.BYTE10</code> bit field contains user bit data for byte 10 of sub frame B.
15:8 (R/W)	BYTE9	Byte 9 Sub Frame B. The <code>SPDIF_TX_UBUFF_B2.BYTE9</code> bit field contains user bit data for byte 9 of sub frame B.
7:0 (R/W)	BYTE8	Byte 8 Sub Frame B. The <code>SPDIF_TX_UBUFF_B2.BYTE8</code> bit field contains user bit data for byte 8 of sub frame B.

Transmit User Buffer B3 Register

The `SPDIF_TX_UBUFF_B3` register holds the transmit user buffer data for bytes 12-15 for sub frame B.

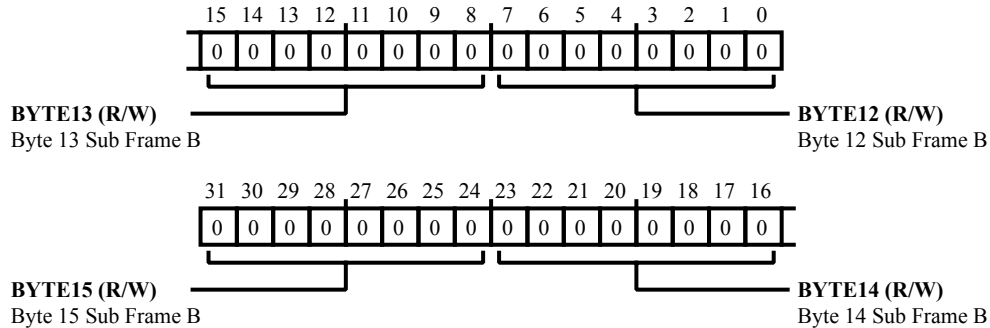


Figure 26-36: SPDIF_TX_UBUFF_B3 Register Diagram

Table 26-33: SPDIF_TX_UBUFF_B3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE15	Byte 15 Sub Frame B. The <code>SPDIF_TX_UBUFF_B3.BYTE15</code> bit field contains user bit data for byte 15 of sub frame B.
23:16 (R/W)	BYTE14	Byte 14 Sub Frame B. The <code>SPDIF_TX_UBUFF_B3.BYTE14</code> bit field contains user bit data for byte 14 of sub frame B.
15:8 (R/W)	BYTE13	Byte 13 Sub Frame B. The <code>SPDIF_TX_UBUFF_B3.BYTE13</code> bit field contains user bit data for byte 13 of sub frame B.
7:0 (R/W)	BYTE12	Byte 12 Sub Frame B. The <code>SPDIF_TX_UBUFF_B3.BYTE12</code> bit field contains user bit data for byte 12 of sub frame B.

Transmit User Buffer B4 Register

The `SPDIF_TX_UBUFF_B4` register holds the transmit user buffer data for bytes 16-19 for sub frame B.

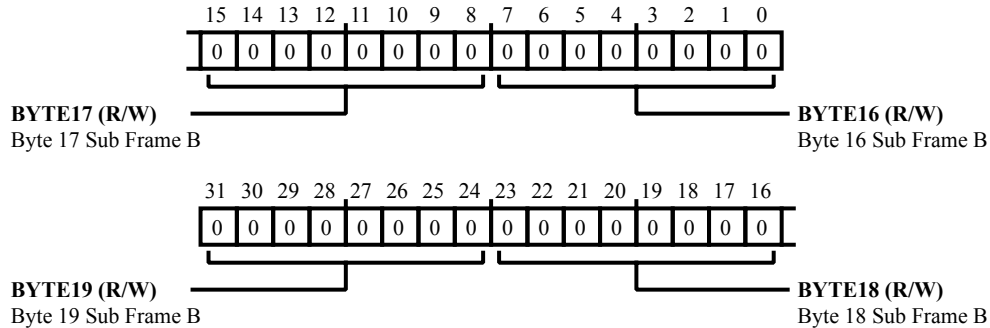


Figure 26-37: SPDIF_TX_UBUFF_B4 Register Diagram

Table 26-34: SPDIF_TX_UBUFF_B4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE19	Byte 19 Sub Frame B. The <code>SPDIF_TX_UBUFF_B4.BYTE19</code> bit field contains user bit data for byte 19 of sub frame B.
23:16 (R/W)	BYTE18	Byte 18 Sub Frame B. The <code>SPDIF_TX_UBUFF_B4.BYTE18</code> bit field contains user bit data for byte 18 of sub frame B.
15:8 (R/W)	BYTE17	Byte 17 Sub Frame B. The <code>SPDIF_TX_UBUFF_B4.BYTE17</code> bit field contains user bit data for byte 17 of sub frame B.
7:0 (R/W)	BYTE16	Byte 16 Sub Frame B. The <code>SPDIF_TX_UBUFF_B4.BYTE16</code> bit field contains user bit data for byte 16 of sub frame B.

Transmit User Buffer B5 Register

The `SPDIF_TX_UBUFF_B5` register holds the transmit user buffer data for bytes 20-23 for sub frame B.

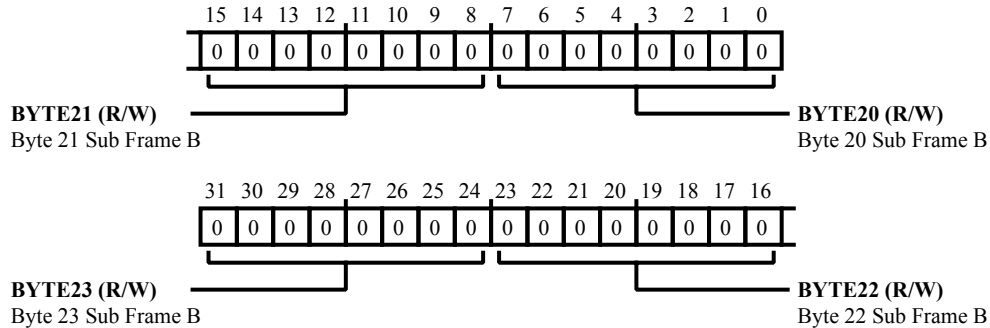


Figure 26-38: SPDIF_TX_UBUFF_B5 Register Diagram

Table 26-35: SPDIF_TX_UBUFF_B5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE23	Byte 23 Sub Frame B. The <code>SPDIF_TX_UBUFF_B5.BYTE23</code> bit field contains user bit data for byte 23 of sub frame B.
23:16 (R/W)	BYTE22	Byte 22 Sub Frame B. The <code>SPDIF_TX_UBUFF_B5.BYTE22</code> bit field contains user bit data for byte 22 of sub frame B.
15:8 (R/W)	BYTE21	Byte 21 Sub Frame B. The <code>SPDIF_TX_UBUFF_B5.BYTE21</code> bit field contains user bit data for byte 21 of sub frame B.
7:0 (R/W)	BYTE20	Byte 20 Sub Frame B. The <code>SPDIF_TX_UBUFF_B5.BYTE20</code> bit field contains user bit data for byte 20 of sub frame B.

User Bit Update Register

After writing to the transmit user buffer registers, a value of 0x1 must be written into the `SPDIF_TX_USRUPDT` register to enable the use of these user buffer bits in the next transfer block.

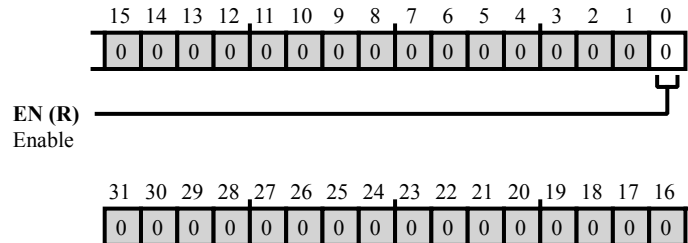


Figure 26-39: SPDIF_TX_USRUPDT Register Diagram

Table 26-36: SPDIF_TX_USRUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	EN	Enable. After writing to the transmit user buffer registers, a value of 0x1 must be written into the <code>SPDIF_TX_USRUPDT</code> register to enable the use of these <code>SPDIF_TX_USRUPDT.EN</code> bits in the next transfer block.

27 Direct Memory Access (DMA)

The processor architecture distributes the DMA channels throughout the infrastructure. Often, the channels cluster together through system crossbars (SCB), sharing a single interface with the main system crossbar.

The DMA channels can perform transfers between memory and a peripheral or between one memory and another memory. Memory-to-memory DMA transfers (MDMA) require two DMA channels. One channel is the source channel, and the second, the destination channel.

All DMA channels can transport data to and from virtually all on-chip and off-chip memories.

DMA transfers on the processor use either a descriptor-based method or register-based method. Register-based DMA allows the processor directly to program DMA controller registers to initiate a DMA transfer. On completion, the controller registers can automatically update with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based transfers allow the chaining together of multiple DMA sequences. In descriptor-based DMA operations, DMA channel programming can automatically set up and start another DMA transfer after the current sequence completes.

The DMA channel does not connect external memories and devices directly. Rather, data passes through an external-memory interface port. DMA operations can access any device the external memory interface supports. These interfaces typically include:

- Flash memory
- SRAM
- FIFOs
- Memory-mapped peripheral devices
- Dynamic Memory (if present)

DMA Channel Features

The processor uses Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure and interface with the system crossbar unit (SCB).

The following is a list of DMA interface features.

- Supports integer byte strides including byte strides of 0 and negative byte strides
- Register-based configuration
 - Core writes DMA configuration
 - Supports automatic reloading for continuous operation
- Flexible descriptor-based configuration
 - DMA descriptors are fetched from memory
 - Support for variable descriptor sizes
- Flexible flow control – Transitions between the various descriptor-based modes and for DMA termination
- Orthogonal transfers
 - Support for three transfer dimensions
 - One and two dimensional (1D and 2D) transfers supported per descriptor set
 - Three dimensional (3D) support provided by chained descriptor sets
- Configurable memory and peripheral-transfer word sizes
 - Memory interface supports 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit transfers
 - Peripheral interface supports for 8-bit, 16-bit, and 32-bit transfers
- Interrupt notification
 - Row or work unit completion
 - Error conditions
- Incoming and outgoing trigger support
 - Trigger generation for row or work unit completion
 - Work unit can wait for incoming trigger
- MMR access bus – Provides access to memory-mapped registers for configuration, monitoring, and debug
- SCB crossbar interface connects the DMA channel to the system crossbar
- Peripheral DMA bus – Interfaces the DMA channel to a peripheral or another DMA channel
- Peripheral data-request interrupt support
- Bandwidth monitoring and limiting for MDMA channels

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

NOTE: There are two types of peripherals that use DMA. The first have dedicated DMA channels controlled by the Dedicated DMA Engine (DDE) and are described in this chapter. The second type is not controlled by the DDE. These peripherals have their own operating modes and programming models (see the peripheral chapter for this information). The complete list of DMA supported peripherals are shown in the [Table 41-4 SCB Controlled DMA Channel Peripherals](#) .

ADSP-2156x DMA Register List

The DMA channel controller (DMA) supports data transfers within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure, as DMA's. A set of registers governs DMA operations. For more information on DMA functionality, see the DMA register descriptions.

Table 27-1: ADSP-2156x DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

ADSP-2156x DMA Channel List

Table 27-2: ADSP-2156x DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA1	SPORT0_B_DMA	SPORT0 Channel B DMA
DMA2	SPORT1_A_DMA	SPORT1 Channel A DMA
DMA3	SPORT1_B_DMA	SPORT1 Channel B DMA
DMA4	SPORT2_A_DMA	SPORT2 Channel A DMA
DMA5	SPORT2_B_DMA	SPORT2 Channel B DMA
DMA6	SPORT3_A_DMA	SPORT3 Channel A DMA
DMA7	SPORT3_B_DMA	SPORT3 Channel B DMA
DMA8	MDMA0_SRC	Memory DMA Stream 0 Source Channel
DMA9	MDMA0_DST	Memory DMA Stream 0 Destination Channel
DMA10	SPORT4_A_DMA	SPORT4 Channel A DMA
DMA11	SPORT4_B_DMA	SPORT4 Channel B DMA
DMA12	SPORT5_A_DMA	SPORT5 Channel A DMA
DMA13	SPORT5_B_DMA	SPORT5 Channel B DMA
DMA14	SPORT6_A_DMA	SPORT6 Channel A DMA
DMA15	SPORT6_B_DMA	SPORT6 Channel B DMA
DMA16	SPORT7_A_DMA	SPORT7 Channel A DMA
DMA17	SPORT7_B_DMA	SPORT7 Channel B DMA
DMA18	MDMA1_SRC	Memory DMA Stream 1 Source Channel
DMA19	MDMA1_DST	Memory DMA Stream 1 Destination Channel
DMA20	UART0_TXDMA	UART0 Transmit DMA
DMA21	UART0_RXDMA	UART0 Receive DMA
DMA22	SPI0_TXDMA	SPI0 TX DMA Channel
DMA23	SPI0_RXDMA	SPI0 RX DMA Channel
DMA24	SPI1_TXDMA	SPI1 TX DMA Channel
DMA25	SPI1_RXDMA	SPI1 RX DMA Channel
DMA26	SPI2_TXDMA	SPI2 TX DMA Channel

Table 27-2: ADSP-2156x DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA27	SPI2_RXDMA	SPI2 RX DMA Channel
DMA30	LP0_DMA	LP0 DMA Channel
DMA34	UART1_TXDMA	UART1 Transmit DMA
DMA35	UART1_RXDMA	UART1 Receive DMA
DMA36	LP1_DMA	LP1 DMA Channel
DMA37	UART2_TXDMA	UART2 Transmit DMA
DMA38	UART2_RXDMA	UART2 Receive DMA
DMA39	MDMA2_SRC	Memory DMA Stream 2 Source Channel
DMA40	MDMA2_DST	Memory DMA Stream 2 Destination Channel
DMA43	MDMA3_SRC	Memory DMA Stream 3 Source Channel
DMA44	MDMA3_DST	Memory DMA Stream 3 Destination Channel

DMA Definitions

To make the best use of the DMA controller, it is useful to understand the following terms.

Descriptor

A single element of a descriptor set that maps to a specific register of a DMA channel.

Descriptor Fetch

The action of retrieving descriptors from memory through memory read operations and loading them into the DMA channel registers upon their read return.

Descriptor Set

An array of descriptors associated with a single work unit. The user can configure the size of the descriptor set. See [Descriptor-Based Flow Modes](#).

Disabled State

The channel is disabled because the enable bit = 0 or as a result of an error.

DMAC

An acronym used for a DMA cluster.

DMA Channel

A single DMA engine that has all the capabilities and registers as defined for a given processor. A DMA channel or engine is connected to a single peripheral.

DMA Cluster

A grouping of multiple DMA channels with a shared SCB crossbar interface, controller, and arbiter. Also known as a DMAC.

Initial Descriptor

The first descriptor in the descriptor set.

MDMA

Memory-to-Memory DMA data transfer. Two DMA channels are paired to perform a memory read from one address location and a memory write of that data to another address location.

Stop State

A time where the channel is enabled but not currently programmed to perform a data transfer. Programming the flow to STOP causes the channel to enter the stop state at the end of the work unit.

User

Any person, debug, emulator, software routine, or action taken by the core that accesses the MMR registers of the DMA channel or peripherals, or sets up data and descriptors in memory.

Wait State

If instructed to wait for a trigger, the channel enters this state once it has completed a work unit. The channel remains in this state until a trigger occurs. If a trigger came in before reaching the wait state, the channel skips over the wait state upon completion of the work unit.

Work Unit

A single data transaction or series of data transactions performed based on the configuration of the DMA channel. For autobuffer mode, a new work unit is defined at the time all current count registers are initialized to start values. Once all the current count registers count down to zero, the work unit has completed.

Work Unit Chain

A single work unit or a series of work units separated by a `STOP` or disabled state. The work units in the chain (except the last one) are programmed to the required descriptor flow. The last work unit in the chain is programmed to a flow of `STOP` or `AUTO`. `STOP` terminates the state at the end of that work unit. `AUTO` must be terminated by disabling the DMA channel. A work unit chain is also known as a descriptor chain.

Block Diagram

The *DMA Channel Block Diagram* shows the functional blocks within the DMA interface.

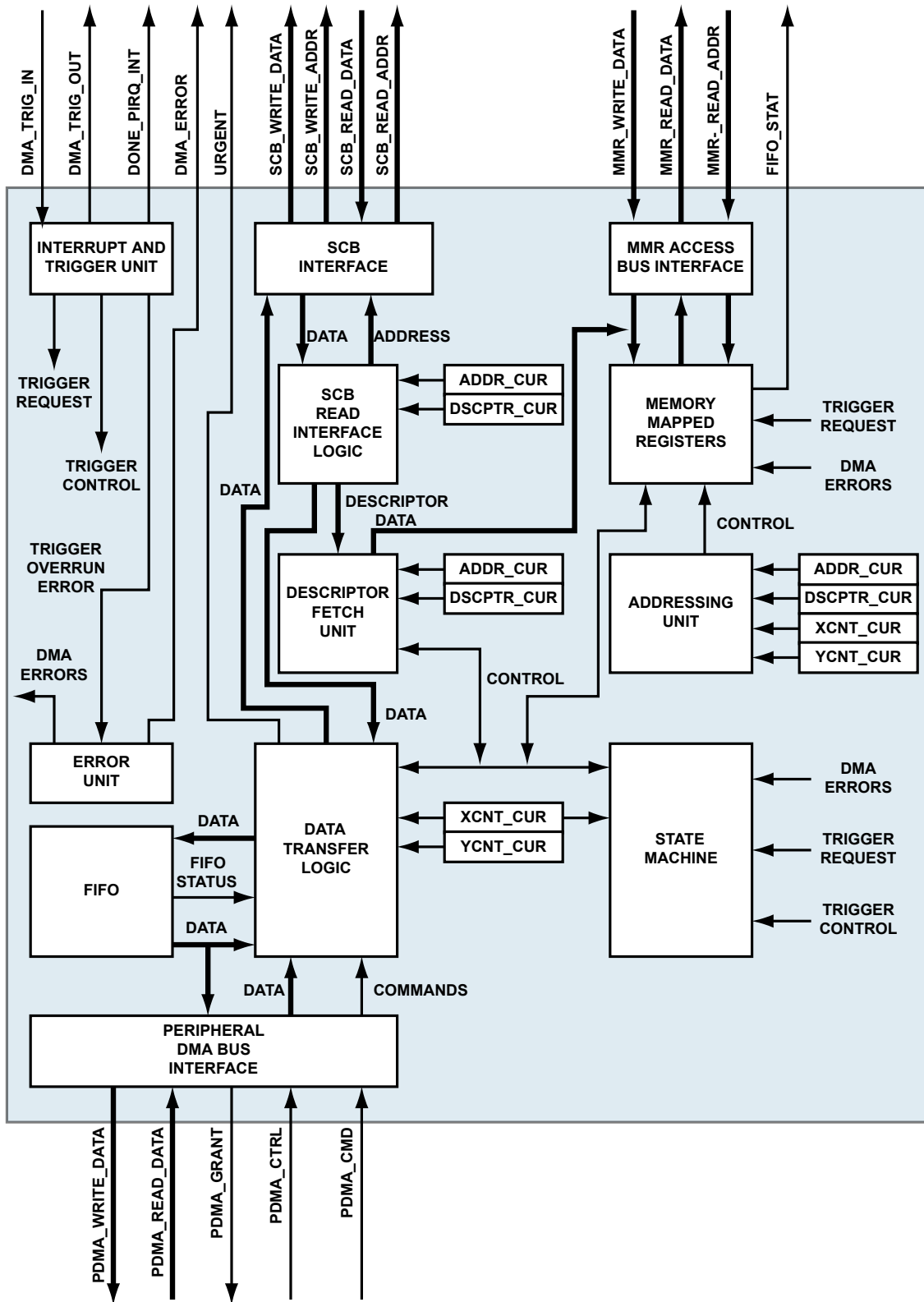


Figure 27-1: DMA Channel Block Diagram

For more information on the interfaces, see:

- [DMA Channel Peripheral DMA Bus](#)
- [Medium Band Width DMA Channel MMR Access Bus](#)
- [DMA Channel Event Control](#)
- [DMA Channel SCB Interface](#)

Architectural Concepts

The DMA channel provides a method to transfer data between memory spaces or between memory and a peripheral using a number of system interfaces. The DMA channel provides an efficient method of distributing data throughout the system, freeing up the processor core for other operations. Each peripheral that supports DMA transfers has its own dedicated DMA channel or channels with its own register set. The register set configures and controls the operating modes of the DMA transfers.

DMA Channel SCB Interface

The SCB interface connects the DMA channel to the SCB crossbar allowing for transfers to and from the processors internal memory and other suitable system resources.

The DMA channel connects to the system interconnect through the SCB interface. This connection lets the DMA channel perform work-unit data transfers with memories such as L1, L2 (internal), and L3 (external). In addition to work unit data transfers, the SCB interface also is used for fetching descriptor sets for all the descriptor-based transfer modes.

The DMA channel can support data bus widths of 16, 32, 64, or 128 bits. The data bus widths for a given DMA channel on a specific processor can vary and are not configurable. Read the `DMA_STAT.MBWID` field to determine the assigned bus widths.

SCB Interface Signals

The DMA channel operates at one of the $SCLK_n$ frequencies, as does the SCB interface. $SCLK_0$ clocks all but four DMA channels which are clocked by $SCLK_1$. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance as shown in the *SCB Interface Signals* table.

Table 27-3: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16, 32, 64, or 128	Data bus used for write operations. The width of the bus can be determined from <code>DMA_STAT.MBWID</code> .
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction.
SCB_READ_DATA	16, 32, 64, or 128	Data bus used for read operations. The width of the bus can be determined from <code>DMA_STAT.MBWID</code> .
SCB_READ_ADDRESS	32	Read address bus. Provides the address of the first transfer in a burst transaction.

SCB Burst Transfers

The SCB interface supports burst transfers for memory read and write operations. The burst length is a function of the configurable memory size of the DMA channel for the work unit and the fixed bus width of the SCB data bus of the DMA channel.

- If the DMA channel configuration selects a memory transfer size less than or equal to the DMA channels bus width, the burst length is always 1.
- If the configured memory size is greater than the SCB interface bus width, the burst length is sufficient to transfer a transaction as specified by the configured memory size.

Table 27-4: DMA Channel SCB Burst Lengths

Configured Memory Size	Burst Length			
	<i>16-bit Bus</i>	<i>32-bit Bus</i>	<i>64-Bit Bus</i>	<i>128-bit Bus</i>
1 Byte	1	1	1	1
2 Bytes	1	1	1	1
4 Bytes	2	1	1	1
8 Bytes	4	2	1	1
16 Bytes	8	4	2	1
32 Bytes	16	8	4	2

Data Address Alignment

To prevent addressing errors and to maximize bandwidth of the SCB interface to the DMA channel, data addresses align with a multiple of the programmable memory size of the DMA channels configuration. These configuration options appear in the [Descriptor Set Address Alignment](#) table.

There are situations in which entire work units may not transfer at the maximum configurable memory size. In this case, the entire work unit can transfer by reducing the configured memory size at the expense of bus bandwidth using descriptor sets as follows:

- The first descriptor set can be configured to transfer data until the larger memory size alignments are met.
- A second descriptor set with a larger memory size configuration then can be used to transfer the bulk of the data in the work unit.
- Finally, a third descriptor set can be used with a smaller memory size to complete any final data transfers that cannot meet the alignment requirements of the previous descriptor set configuration.

Table 27-5: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction
2 Bytes	ADDR[0] == 0

Table 27-5: DMA Channel Address Alignment Requirements (Continued)

Configured Memory Size	Address Restriction
4 Bytes	ADDR[1:0] == 0
8 Bytes	ADDR[2:0] == 0
16 Bytes	ADDR[3:0] == 0
32 Bytes	ADDR[4:0] == 0

Descriptor Set Address Alignment

All descriptor set addresses and descriptors within a descriptor set must align to a 32-bit address. For descriptor set fetches, the DMA engine ignores the memory-size configuration of the DMA channel. This feature avoids the need to align descriptor sets based on the memory width configuration of the previous descriptor set.

For descriptor sets containing only a single descriptor, the transfer takes place as a single 32-bit transfer. For descriptor sets containing multiple descriptors, the DMA engine fetches each 32-bit descriptor individually and treats it as multiple 32-bit transfers.

Peripheral Control Commands

The peripheral DMA bus of the DMA channel provides a means for peripherals on the processor to issue commands to the DMA channel. These commands provide greater control over the DMA channel operation. This control improves real-time performance and relieves control and interrupt demands on the core. Peripherals can send commands to the DMA controller over the 3-bit PERI_CMD bus. The DMA control commands extend the set of operations available to the peripheral beyond the simple “request data” command used by peripherals in general. Refer to the appropriate peripheral chapter for a description on how that peripheral uses DMA control commands.

These DMA control commands (see the *PDMA_CMD Peripheral DMA Control Commands* table) are not visible to or controlled by the program. But, their use by a peripheral has implications for the structure of the DMA transfers that the peripheral can support. It is important to write application software such that it complies with certain restrictions, regarding work units and descriptor chains. Complying with this guideline makes the peripheral operate properly whenever it issues DMA control commands.

The *PDMA_CMD Peripheral DMA Control Commands* table describes the commands the DMA controller issues. The following sections describe these commands in more detail.

Table 27-6: PDMA_CMD Peripheral DMA Control Commands

Command	Name	Description
b#000	NOP	No operation
b#001	Restart	Restarts the current work unit from the beginning
b#010	Finish	Finishes the current work unit and starts the next
b#011	Interrupt	Immediately sets the DMA completion interrupt in the DMA channel

Table 27-6: PDMA_CMD Peripheral DMA Control Commands (Continued)

Command	Name	Description
b#100	Request Data	Typical DMA data request
b#101	Request Data Urgent	Urgent DMA data request
b#110	Reserved	Reserved
b#111	Reserved	Reserved

Idle Command

The DMA channel drives this command when the enabled peripheral has no data requests required.

Request-Data Command

The request data command is a request for data transfers between the DMA channel and the peripheral. The request is held by the peripheral until granted or acknowledged by the DMA channel.

Request-Data Urgent Command

The request-data urgent command behaves identically to the request data command, except that---during the commands assertion---the DMA channel performs its memory accesses with urgent priority. This priority includes both data and descriptor fetch memory accesses. For example, a DMA management capable peripheral can use this control command if an internal FIFO approaches a critical condition.

The request is held by the peripheral until granted or acknowledged by the DMA channel.

Peripheral-Control Command Restrictions

The proper operation of the DMA channel FIFO leads to certain restrictions in the sequence of DMA peripheral control commands issued by a peripheral. The following sections describe these restrictions.

DMA Channel Peripheral DMA Bus

The DMA channel connects to peripherals or other DMA channels through the peripheral DMA bus. This bus is a dedicated point-to-point interface supporting data bus widths of 8, 16, 32, or 64 bits. The data bus widths for a given DMA channel on a particular processor can vary and are not configurable. Reading the `DMA_STAT.PBWID` field permits determining the assigned bus width.

The DMA channel operates at one of the SCLK frequencies, as does the peripheral DMA bus. The *Peripheral DMA Bus Signals* table provides descriptions of the peripheral DMA bus signals.

Table 27-7: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8, 16, 32, or 64	Data bus used for write operations. The width of the bus can be determined from <code>DMA_STAT.PBWID</code> .
PDMA_READ_DATA	8, 16, 32, or 64	Data bus used for read operations. The width of the bus can be determined from <code>DMA_STAT.PBWID</code> .

Table 27-7: Peripheral DMA Bus Signals (Continued)

Signal	Width (bits)	Description
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit). These signals indicate that the DMA channel is ready to receive data for write operations (peripheral receive).
PDMA_CMD	3	The peripheral uses the signal for issuing DMA channel control commands.
PDMA_CTRL		The peripheral uses the control signals to send various commands to the DMA channel and control the direction of flow.

Memory DMA and Triggering

A memory DMA (MDMA) channel provides a means of doing memory-to-memory DMA transfers among the various memory spaces that have DMA support.

The DMA controller implements memory DMA (MDMA) channels by interfacing two DMA channels through the peripheral DMA bus interface. One DMA channel serves for memory read operations, and the second channel serves for memory writes. Depending on the processor, a memory DMA channel can have an additional peripheral, such as a CRC peripheral. The additional peripheral is inserted into the peripheral DMA bus that optionally can be enabled.

MDMA channel configurations that do not involve an additional peripheral impose no restrictions on which of the DMA channels is used for the read operation or the write operation. But, the configuration of both channels cannot have the same transfer direction. For MDMA channel configurations that enable a peripheral between the read and write channels, be aware of possible restrictions imposed on which channel can be used for a given transfer direction.

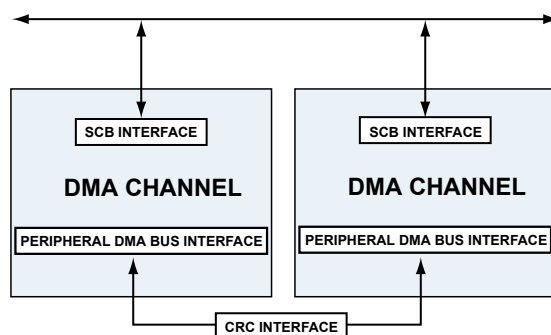


Figure 27-2: MDMA Channel Dedicated Pair

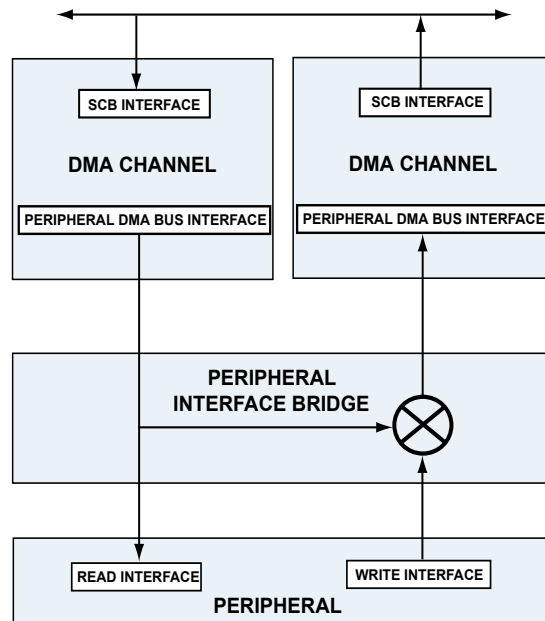


Figure 27-3: MDMA Channel Pair with Peripheral

A memory-to-memory transfer always requires enabled source and destination channels. Because the channels interface through the peripheral DMA bus and can have an additional peripheral inserted into the peripheral DMA bus, programs must make sure to set the same values in the `DMA_CFG.PSIZE` of both the source and destination channels.

The memory DMA channels support the full range of the `DMA_CFG.MSIZE` options for the DMA transfers to and from the memories.

Because the MDMA channel consists of two DMA channels, the entire MDMA channel has two sets of FIFOs, one in the read channel and one in the write channel. This FIFO usage allows for more efficient bursting of both read and write transactions using the available bandwidth. While the `DMA_CFG.PSIZE` configuration must be identical for both source and destination DMA channels, this restriction does not apply for the `DMA_CFG.MSIZE` configuration.

Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus. From a performance perspective, use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `PSIZE` value used.

NOTE: This is applicable for all of the DMA channels, except the MDMA channels 2 and 3. For the MDMA 2 and 3, the minimum `PSIZE` and `MSIZE` is 4 bytes (32 bits).

The independent source and destination DMA channels also have their own dedicated interrupt and trigger events. While it is normal practice to have only event generation performed at destination DMA completion, programs also can use other means of interrupt generation.

Configuration of an MDMA transfer is done in a similar manner to peripheral DMA transfers, except for writing two DMA channel registers instead of one.

To control the pace of data transfers, use triggers on either the memory read or the memory write channel pair used in an MDMA operation. Setting the `DMA_CFG.TWAIT` bit in the memory read channel prevents both channels from transferring data before the system is ready. However, only configuring the memory write channel to wait for a trigger allows for data fetch from the memory in anticipation of the memory write operation.

The processor supports these categories of MDMA channels:

Enhanced Bandwidth MDMA. The channels run in SYSCLK domain and support 32-bit memory and peripheral bus width. The maximum theoretical bandwidth of this MDMA stream is up to $4 \times 500 = 2000$ MB/s.

Maximum Bandwidth MDMA. The channels run in SYSCLK domain and support 64-bit memory and peripheral bus width. The maximum theoretical bandwidth of these MDMA streams is up to $8 \times 500 = 4000$ MB/s.

Medium Band Width DMA Channel MMR Access Bus

The MMR access bus provides access to all the DMA channels memory-mapped registers for DMA channel configuration, monitoring, and debug. The interface has a fixed 32-bit data bus for read and write accesses.

The *MMR Access Bus Signals* table provides descriptions of the MMR access bus signals.

Table 27-8: MMR Access Bus Signals

Signal	Width (bits)	Description
MMR_WRITE_DATA	32	Data bus used for write operations to the MMRs from the core
MMR_READ_DATA	32	Data bus used to return read data from the MMRs
MMR_READ_ADDR	7	Address used to select the MMR to access

DMA Channel Operation Flow

A detailed description of the flow of operation of the DMA channel appears in the following topics:

- [Startup Flow](#)
- [Refresh Flow](#)
- [DMA Operating Modes](#)
- [Stop Mode](#)
- [DMA Channel Errors](#)

Startup Flow

Enabling a DMA operation on a given channel first requires directly writing some or all of the DMA parameter registers. The minimum set of register required to be initialized depends on the desired mode of operation as described in the following sections.

Startup Minimum-Enable Requirements

To start a DMA operation on a given channel, some or all of the DMA parameter registers must first be initialized and configured to the desired DMA channels operating mode.

- For descriptor-array-based flow modes, at minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For descriptor-list-based flow modes, at minimum, write the `DMA_DSCPTR_NXT` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For non-descriptor-based flow modes, write the `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_XMOD` registers prior to writing the `DMA_CFG` register.

Programs can write other registers that can remain static throughout the course of the DMA activity. The write to the `DMA_CFG` register begins the DMA operation.

ATTENTION: When software directly writes the `DMA_CFG` register, the DMA controller recognizes this action as the special startup condition. This condition occurs when starting the DMA controller for the first time on this channel or occurs after the DMA channel stops. It is possible for the channel to flag a DMA error condition regardless of the `DMA_CFG.EN` bit setting.

Startup Operation

The startup operation is initiated by software directly writing the `DMA_CFG` register when starting DMA for the first time on a channel or after the channel has entered to the stop state.

When the descriptor fetch is complete and the DMA channel is enabled, the `DMA_CFG` descriptor element replaces the `DMA_CFG` register content and assumes control. Before this point, the direct write to the `DMA_CFG` register had control.

At startup, the selected flow mode and the descriptor size determine the course of the DMA initialization process. The `DMA_CFG.FLOW` field determines whether to load more current registers from descriptor sets in memory. The `DMA_CFG.NDSIZE` field details how many descriptor elements to fetch before starting the DMA operation. This process does not affect DMA registers that are not in the descriptor; no modifications are made to their prior values.

For descriptor-list flow modes, the channel copies the `DMA_DSCPTR_NXT` register value into the `DMA_DSCPTR_CUR` register. Then, the channel fetches new descriptor elements from memory. The `DMA_DSCPTR_CUR` register indexes each fetch, and the channel increments the index after each fetch. After completion of the descriptor fetch, the `DMA_DSCPTR_CUR` register points to the next 32-bit word in memory past the end of the descriptor.

If the descriptor fetch is for a descriptor-array mode transfer, the channel does *not* copy the `DMA_DSCPTR_NXT` register into the `DMA_DSCPTR_CUR` register. *Instead*, the descriptor fetch indexing begins with the value in the `DMA_DSCPTR_CUR` register.

If `DMA_CFG` is not part of the fetched descriptor set, the previous value (originally as written on startup) controls the work unit operation. If the `DMA_CFG` register is part of the fetched descriptor set, the value programmed by the

MMR access controls only the loading of the first descriptor set fetched from memory. The configuration of the `DMA_CFG` register controls the subsequent DMA work units of the fetched descriptor set.

After the descriptor fetch is complete or if the flow configuration was originally for one of the register-based flow modes, the DMA operation begins. The DMA channel immediately fills its FIFO. For a memory-write operation, the DMA channel begins accepting data from the peripheral. For a memory-read operation, the DMA channel begins memory reads when the SCB bus grants access to the DMA channel.

When the DMA channel performs its first data-memory access, its address and count computations take their input operands from the start registers. These registers can include `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_YCNT`, if necessary. The channel writes results back to the current registers. These registers include `DMA_ADDR_CUR`, `DMA_XCNT_CUR`, and `DMA_YCNT_CUR`. Note that the current registers are not valid until the channel performs the first memory access, which can be some time after the write to the `DMA_CFG` register starts the channel. Once started, the channel automatically loads the current registers from the appropriate descriptor elements, overwriting their previous contents. These automatic-load operations include:

- The channel copies the `DMA_ADDRSTART` value to `DMA_ADDR_CUR`.
- The channel copies the `DMA_XCNT` value to `DMA_XCNT_CUR`.
- The channel copies the `DMA_YCNT` to `DMA_YCNT_CUR`.

Refresh Flow

When the channel completes processing of a work unit, the DMA channel performs the following operations:

- Completes the transfer of all data between memory and the DMA channel.
- Performs a synchronized transition (if the DMA channel configuration is a memory read operation with the `DMA_CFG.SYNC` bit enabled) *and* transfers all data to the peripheral before continuing.
- Forwards the signals from the DMA channel (if interrupts or triggers are enabled) *and* updates the `DMA_STAT` register to indicate the interrupt request or trigger events.
- Clears the `DMA_STAT.RUN` bit field to stop DMA operation (if the flow was set to stop mode) *and* transfers any remaining data in the FIFO of the DMA channel to the peripheral.
- Loads a new descriptor from memory into the DMA registers by way of the contents of the `DMA_DSCPTR_CUR` register (for descriptor-array mode) *and* increments the `DMA_DSCPTR_CUR` register

The channel takes the descriptor size from the `DMA_CFG.NDSIZE` value before the fetch.

- Copies the `DMA_DSCPTR_NXT` register into the `DMA_DSCPTR_CUR` register (for descriptor-list mode), fetches the descriptor from the new contents of the `DMA_DSCPTR_CUR` register, *and* places these contents into the DMA registers while incrementing the `DMA_DSCPTR_CUR` register.
- Checks for detection of an incoming trigger event (for descriptor-on-demand array mode):
 - If the channel detects a trigger event, the DMA channel loads a new descriptor from memory into the DMA registers from the contents of the `DMA_DSCPTR_CUR` register, while incrementing the

`DMA_DSCPTR_CUR` register. The channel takes the descriptor size from the `DMA_CFG.NDSIZE` value before the fetch.

- If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers.
- Checks for detection of an incoming trigger event (for descriptor-on-demand list mode):
 - If the channel detects a trigger event, the DMA channel copies the `DMA_DSCPTR_NXT` register value to the `DMA_DSCPTR_CUR` register, fetches the descriptor memory from the `DMA_DSCPTR_CUR` register, *and* places the contents into the DMA registers while incrementing the `DMA_DSCPTR_CUR` register.
 - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers as described in the next step.
- Begins the next work unit (if flow configuration is anything other than stop mode) by reloading the current registers (`DMA_ADDR_CUR`, `DMA_XCNT_CUR`, and `DMA_YCNT_CUR`) from their descriptor registers (`DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_YCNT`)

Work Unit Transition Flow

The `DMA_CFG.SYNC` bit controls transitions from one work unit to the next work unit. In general, continuous transitions have lower latency at the cost of restrictions on changes of data format or addressed memory space in the two work units. These latency gains and data restrictions arise from the way the channel handles the DMA FIFO while fetching the next descriptor.

In continuous transitions, with disabled synchronization, the DMA FIFO pipeline continues to transfer data to and from the peripheral or destination memory. These transfers continue during the descriptor fetch and during the DMA channel pause between descriptor chains. By comparison, synchronized transitions provide better real-time synchronization of interrupts and triggers with a given peripheral state. Synchronized transitions also provide greater flexibility in the data formats and memory spaces of the two work units. This flexibility comes at the cost of higher latency in the transition. In synchronized transitions, the DMA FIFO pipeline drains to the destination or flushes (received data discarded) between work units.

NOTE: The `DMA_CFG.SYNC` bit of the MDMA source channel controls work unit transitions for MDMA streams. Clear this reserved bit of the MDMA destination channel, placing it in the disabled state. In transmit (memory read) channels, the `DMA_CFG.SYNC` bit of the last descriptor before the transition controls the transition behavior. In contrast, in receive channels, the `DMA_CFG.SYNC` bit of the first descriptor of the next descriptor chain controls the transition.

Work Unit Transmit and MDMA Source Transitions

In DMA transmit (memory read) and MDMA source channels, the `DMA_CFG.SYNC` bit controls the interrupt timing at the end of the work unit. This bit also controls the handling of the DMA FIFO between the current and the next work unit.

If the `DMA_CFG.SYNC` bit configuration disables synchronization, the DMA channel operates in continuous transition. In a continuous transition, just after reading the last data item from memory, the DMA channel starts all of the following operations parallel:

- Signals the interrupt request or trigger
- Updates the `DMA_STAT` register to indicate DMA completion status
- Begins fetching the next descriptor
- Delivers the final data items from the DMA FIFO to the destination memory or peripheral

This process lets the DMA channel provide data from the FIFO to the peripheral continuously during the descriptor fetch latency period.

If the configuration disables synchronization, the final interrupt request or trigger (if enabled) occurs when the channel reads the last data from memory. This event occurs at the earliest time that the channel safely can modify the output memory buffer without affecting the previous data transmission. There can be a number of data items remaining in the FIFO and not yet at the peripheral. This number depends on the FIFO depth of the DMA channel. In this configuration, do not use the DMA interrupt request as the sole means of synchronizing the shutdown or reconfiguration of the peripheral following a transmission.

NOTE: If the configuration selects continuous transition on a transmit (memory read) descriptor, the next descriptor must have the same:

- Peripheral transfer size (`DMA_CFG.PSIZE`)
- Read or write direction
- Source memory (internal versus external) as the current descriptor

It is possible to disable synchronization by selecting continuous transition on a work unit with configuration for stop-flow mode and with enabled interrupts or triggers. This approach can result in the execution of the event service routine while draining of the final data is ongoing from the FIFO to the peripheral. If data transfers are in-progress, the FIFO is not yet empty. The `DMA_STAT.RUN` bits of the DMA channels indicate this status. Do not start a new work unit with a different peripheral transfer size or direction while data transfers are in-progress.

CAUTION: Disabling the channel with the `DMA_CFG.EN` bit while data transfers are in-progress causes the loss of the data in the FIFO.

A synchronized transition configuration directs the channel to drain the DMA FIFO to the destination memory or peripheral. This FIFO operation occurs before the channel signals any interrupt and before the channel fetches any subsequent descriptor or data. This operation incurs greater latency, but provides direct synchronization between the DMA interrupt and the state of the data at the peripheral.

If the configuration enables synchronization and enables interrupts, on the last descriptor in a work unit, the interrupt occurs when the channel transfers the final data to the peripheral. This event allows the service routine to switch properly to non-DMA transmit operation. When the event vectors to the interrupt service routine, the DMA channel FIFO is empty, and the DMA channel is no longer running (indicated by the `DMA_STAT.RUN` bits).

A synchronized transition also allows greater flexibility in the format of the DMA descriptor chain. When enabled, the next descriptor can have any `DMA_CFG.PSIZE` configuration or read/write direction supported by the peripheral and can come from either memory space (internal or external). This feature can be useful in managing MDMA work unit queues, since it is no longer necessary to interrupt the queue between dissimilar work units.

Work Unit Receive and MDMA Destination Transitions

In DMA receive channels (memory write operations), the `DMA_CFG.SYNC` bit controls the handling of the DMA FIFO between descriptor chains (not individual descriptor sets), during the DMA channel pause. The DMA channel pauses after the descriptor sets configured with stop flow mode are complete. Restart the channel (for example, after an interrupt) by writing the `DMA_CFG` register of the channel with a value that enables the DMA channel. If the configuration disables synchronization in the `DMA_CFG` value of the new work unit, the configuration selects a continuous transition. In this mode, the DMA FIFO retains any data items received during the channel pause, and they are the first items written to memory in the new work unit. This mode of operation provides lower latency at work unit transitions and ensures no dropping of data items during a DMA pause. The channel provides this operation at the cost of certain restrictions on the DMA descriptors.

NOTE: If the `DMA_CFG.SYNC` bit disables synchronization on the first descriptor of a chain after a DMA pause, do not change the configuration of the `DMA_CFG.PSIZE` field of the new chain from the previous descriptor chain (active before the pause). This restriction applies unless the DMA channel is reset between chains by disabling and then re-enabling the DMA channel.

If the `DMA_CFG.SYNC` bit configuration enables synchronization, the channel uses a synchronized transition. In this mode, only the data that the DMA channel receives from the peripheral after the write to the `DMA_CFG` register gets to memory. The channel discards any prior data items transferred from the peripheral to the DMA FIFO before this register write occurs. This operation provides direct synchronization between the data stream received from the peripheral and the timing of the channel restart, which occurs on the write to the `DMA_CFG` register.

For receive DMA operations, the synchronization has no effect in transitions between work units in the same descriptor chain. When the flow mode of previous descriptor was not stopped, the DMA channel did not pause.

If a descriptor chain begins with synchronization enabled, there is no restriction on the `DMA_CFG.PSIZE` of the new chain in comparison with the previous chain.

NOTE: The peripheral transfer size (`DMA_CFG.PSIZE`) must not change between one descriptor and the next in any DMA receive (memory write) channel within a single descriptor chain, regardless of the `DMA_CFG.SYNC` bit setting. In other words, all memory write descriptor sets in a descriptor chain must have the same `DMA_CFG.PSIZE` value. For any DMA receive channel (memory write operation), there is no restriction on changes of peripheral transfer size (internal versus external) between descriptors or descriptor chains.

Transfer Termination and Shutdown Flow

This section describes channel transfer termination and shutdown in stop flow mode and in autobuffer flow mode.

Stop Flow Mode

In stop flow mode, the DMA channel stops automatically after the work unit is complete. If using a list or array of descriptors to control DMA transfers and if every descriptor contains a `DMA_CFG` descriptor element, configure the flow of the final `DMA_CFG` descriptor element to stop mode, stopping the channel gracefully. After completion, the DMA channel remains in the stop state. Do not confuse this state with the disabled state, which either occurs due to a DMA error or occurs through disabling the DMA channel by configuring the `DMA_CFG.EN` bit.

The intention of disabling the DMA channel through a write to the `DMA_CFG.EN` bit is to shut down the DMA channel and to enter the disabled state. All memory and peripheral data transfers cease, and only peripheral interrupts pass through the DMA channels interrupt signals. However, the DMA channel maintains the `DMA_STAT.RUN` bits. For a write to memory, the outstanding memory-transaction counter tracks returning memory write acknowledgments and updates as required.

For memory reads, the outstanding memory-transaction count also tracks returning memory reads. The channel does not write the memory reads into the FIFO. The channel updates the counter to reflect the completion of the transaction, but the channel ignores the data. The `DMA_STAT.RUN` bits remain in the *waiting for write ACK or FIFO drain to peripheral* state and do not change to *stop or idle state* until the return of all outstanding transactions.

When the `DMA_CFG.EN` bit again enables the DMA channel, the channel performs a full reset and clears all counters. If an outstanding memory transaction returns an acknowledgment or read data after this event, a memory transaction error occurred, which generates an error event. Programs must ensure that all outstanding memory transactions complete before reconfiguring the DMA channel. For example, programs can poll the `DMA_STAT.RUN` bits to return to the stop or idle state before proceeding.

Autobuffer Flow Mode

In this mode, the flow does not use any descriptors in stored memory. Instead, the channel performs DMA in a continuous circular buffer fashion, based on user-programmed DMA register settings. On completion of the work unit, the channel reloads the parameter registers into the current registers, and the DMA controller resumes immediately with zero overhead. Consider this mode as a succession of automatically restarted work units.

For autobuffer-flow modes, the only way to cease operations is to disable the DMA channel through the `DMA_CFG.EN` bit. One method of changing to a new work unit is:

- Disable the DMA channel
- Set up all the registers (and descriptors in memory, if used) except for `DMA_CFG`
- Poll `DMA_STAT.RUN` to wait for the status to reflect stop or idle state, and
- Write `DMA_CFG` to the new configuration to begin the next work unit

In autobuffer-flow mode or for a list or array of descriptor sets without `DMA_CFG` descriptors, use an MMR write to the `DMA_CFG` register to terminate the DMA transfer process. Configure the value of the `DMA_CFG.EN` bit in this register to disable the DMA channel.

CAUTION: When the configuration disables a DMA channel, the DMA controller disables interrupt logic that is based on work unit transitions. Be aware of the system environment and current actions, so that additional interrupts are not required from the DMA channel.

CAUTION: If disabled through `DMA_CFG.EN` in the middle of a transaction, the DMA channel completes any transactions that have begun and avoids generating bus errors. However, the channel considers the action of re-enabling the DMA as a hard reset for all internal DMA channel components. Therefore, pay attention to that particular action to avoid unexpected results.

DMA Channel Errors

When an error occurs, the DMA channel maintains all the state and register values that allow programs to diagnose error causes more thoroughly. The greatest benefit to the programmer is to know exactly what operational state the DMA channel was in at the exact moment the error occurred.

Take care to address the root cause of the error, whether or not the problem originated in the DMA channel. If not properly resolved, the error can result in an additional error shortly after operations resume. The problem can cause other errors elsewhere in the DMA channel or associated modules and circuitry. So, take care also to address those potential problems. Ensure that all outstanding memory reads and writes are complete or cleared before resuming DMA channel operation.

After addressing all issues and neutralizing all side effects of any errors, clear the `DMA_STAT.ERRC` status field and restart the DMA channel by disabling then re-enabling the DMA channel through the `DMA_CFG.EN` bit.

The following sections describe the error types.

Status and Debug Errors

DMA channel error conditions can cause the DMA process to end abnormally. The DMA channel provides error detection as a tool for system development and debug, helping to identify DMA-related programming errors. When the DMA channel detects an error, the channel immediately stops and discards any returned memory-read transactions. The `DMA_STAT.RUN` field of the DMA channel indicates the idle state after acknowledging all outstanding memory transactions. In addition, the channel asserts an error interrupt request and updates the `DMA_STAT.IRQERR` field. Also, the channel updates the `DMA_STAT.ERRC` field, indicating the error cause of the first detected error. Unless the error occurs at the exact moment that modification of register values occurs, the registers contain the error values.

All the DMA error interrupt requests are combined into a single shared interrupt request output. Combined error signals require reading the `DMA_STAT` register of each DMA channel associated with a combined error interrupt request to determine the DMA channel responsible for the generation of the interrupt.

The DMA channel error interrupt handler performs the following actions:

- Read the `DMA_STAT` register of each DMA channel, seeking a channel with the `DMA_STAT.IRQERR` set to indicate an error.
- Read the `DMA_STAT.ERRC` field of each DMA channel, determining the cause of the error.
- Clear the problem with the DMA channel. For example, fix the register values.

- Clear the error in the DMA channel through a write-1-to-clear operation to the `DMA_STAT.IRQERR` bit.

If the channel flags any uncleared error other than a bandwidth monitor error, the channel reports no other error. If the channel reports an uncleared bandwidth monitor error, the channel reports any newly detected error through updating the `DMA_STAT.ERRC` field.

DMA Configuration Register Errors

The channel only flags these configuration errors when the `DMA_CFG.EN` bit enables the DMA channel. Error flagging occurs when the configuration:

- Uses a reserved setting
- Enables `DMA_CFG.TWAIT` in descriptor on-demand flow mode
- Uses an illegal `DMA_CFG.NDSIZE`
- Uses an illegal `DMA_CFG.MSIZE`
- Configures `DMA_XCNT = 0` or, when `DMA_YCNT = 0` in 2D DMA mode
- Uses non-zero value in `DMA_CFG.NDSIZE` when DMA is configured in stop mode or auto mode
- Enables interrupt or outgoing triggers on `DMA_YCNT` when DMA is configured in 1D mode
- Use a `DMA_CFG.MSIZE` that exceeds the FIFO size of the DMA channel
- Uses an illegal `DMA_CFG.PSIZE`
- Uses a `DMA_CFG.PSIZE` that exceeds the FIFO size
- Uses a `DMA_CFG.PSIZE` that exceeds the bus width
- Attempts to change from a transmit operation (memory read) to a receive operation without properly syncing in the previous work unit or when it is the first work unit in a new chain
- Attempts to change `DMA_CFG.PSIZE` of a transmit operation (memory read) without properly syncing in previous work unit or when it is the first work unit in a new chain
- Attempts to change from receive operation (memory write) to a transmit operation during a descriptor chain.
The channel only can change from receive to transmit if the new transmit is synchronized and is the first work unit.
- Attempts to change `DMA_CFG.PSIZE` of a receive operation (memory write) when the operation was not the first work unit (with `DMA_CFG.SYNC` enabled)

Illegal Register Write During Run

The channel generates an error when a write occurs to writable registers of an enabled, running DMA channel. The channel blocks the write. The `DMA_STAT`, `DMA_BWLCNT`, and `DMA_BWMCNT` registers are exempt from this behavior.

Address Alignment Error

The channel generates an address alignment error when any of the following apply:

- Alignment of a descriptor address is not on a 32-bit boundary.
- The current `DMA_CFG.MSIZE` configuration contains an unaligned transfer address. The `DMA_ADDRSTART` register is not aligned according to the `DMA_CFG.MSIZE` field.

Memory Access Error

The channel generates a memory access error when the DMA process:

- attempts to access an unpopulated address,
- attempts to access a location that provokes a security violation

The error returned from the memory triggers the memory access error.

Trigger Overrun Error

A trigger overrun error is generated when a new trigger input occurred while an outstanding trigger is waiting. This error is only generated if `DMA_CFG.TOVEN` is enabled.

Bandwidth-Monitor Error

The channel generates this error when the bandwidth-monitor count expires. This error is not fatal, and the DMA channel continues operation.

Control Interface Error

The channel reports control-interface errors as bus errors to the bus master. This error can result from:

- An address error
- A register write error (write to a read-only register)

DMA Operating Modes

The DMA channel supports a number of different flow modes that control how the DMA channel progresses from one work unit to the next.

The flow mode of a DMA channel is not a global setting. A DMA descriptor set can include the descriptor responsible for configuring the flow of the work unit. There is no restriction, limiting the flow configuration to be the same for the entire descriptor chain. If the descriptor chain is not endless, the last descriptor set configures the flow to stop mode, which results in termination of the descriptor chain after the work unit completes. Another example for mixing flow modes is to create an endless descriptor-array. The configuration of the last descriptor set in the array selects the descriptor-list mode. The next descriptor pointer in this set of descriptors points to the first descriptor in the array.

Register-Based Flow Modes

Register-based DMA operations require configuration by directly writing to the memory-mapped registers of the DMA channel.

Register-based DMA is the traditional method of DMA operation. Software writes all of the configuration of the DMA channel into the memory-mapped registers. This configuration includes information such as the source or destination address and length of the data in the transfer. The DMA controller then starts channel operation. The DMA channel supports the following register-based flow modes.

- [Stop Mode](#)
- [Autobuffer Mode](#)

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor. The supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to the descriptor set size being configurable, descriptor-based DMA also allows altering the flow mode of the next descriptor set. This feature allows for the transition from descriptor-array mode to descriptor-list mode and permits configuring the flow to stop or autobuffer mode.

Stop Mode

In stop mode, the DMA operation executes only once. If started, the DMA channel transfers the desired number of data words and stops itself again when finished. If the DMA channel is no longer used, software configures the enable bit to disable a paused channel. The channel also can generate interrupts and triggers for each row or work unit completion, depending on the desired operation.

Autobuffer Mode

In autobuffer mode, the DMA operates repeatedly in a circular manner. If the transfer of all data words completes, the channel reloads the address pointer ([DMA_ADDR_CUR](#)) automatically with the [DMA_ADDRSTART](#) value. The channel also can generate an interrupt.

The `DMA_CFG.FLOW` field enables autobuffer mode. The configuration must load the `DMA_CFG.NDSIZE` field value, such that the next descriptor size is zero.

Descriptor-Based Flow Modes

Descriptor-based DMA operations fetch descriptor sets from memory allowing for autonomous loading of work units on other work units. Software does not need to set up the DMA sequences directly by writing into the DMA controller registers. Rather, software keeps DMA descriptor sets in memory.

Descriptor-based DMA operations have the following additional attributes.

- The DMA controller autonomously loads the descriptor set from memory to the affected DMA controller registers on demand.
- The channel can fetch descriptor sets from any memory space that supports DMA read operations.
- The descriptor set describes the next operation that the DMA controller performs.

- The descriptor set can include information such as the DMA configuration word as well as data source or destination address, transfer count, and address modify values.

A descriptor set describes a single work unit. The next work unit can reuse some values from the previous one descriptor set. But, this reuse is possible only if they are not overwritten in the subsequent descriptor set fetches and only if the work unit requires the use of this descriptor.

The DMA channel supports the following flow modes with descriptor-based operations.

- [Descriptor-Array Mode](#)
- [Descriptor-List Mode](#)
- [Descriptor-On-Demand Modes](#)

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor and the supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to configurable descriptor set size, descriptor-based DMA also allows for altering of the flow mode of the next descriptor set. Programs can transition from one descriptor-based mode to another descriptor-based mode and can also transition to any of the register-based flow modes.

Descriptor-Array Mode

When configured in this mode, the descriptor sets do not contain further descriptor pointers. Software writes the initial descriptor-pointer value, which points to an array of descriptors. This operation assumes that the individual descriptors reside next to each other and assumes that their addresses are known.

The *Offsets for Descriptor-Array Mode Parameters and Descriptors* table illustrates how to structure a descriptor set in memory. The descriptor sets must reside in a contiguous block of memory in the format shown in the table. Locate the first descriptor of the next descriptor set in the memory location immediately following the last descriptor of the current descriptor set. The values have the same order as the corresponding offset addresses of the memory-mapped register.

Table 27-9: Offsets for Descriptor -Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current registers between the descriptor set fetch and the start of the DMA operation for the work unit.

NOTE: At a minimum, write the [DMA_DSCPTR_CUR](#) register prior to writing to the [DMA_CFG](#) register, which is the special action required to start the DMA channel.

Descriptor-List Mode

In this flow mode, multiple descriptors form a chained list in which each descriptor set contains a pointer to the next descriptor set, allowing greater flexibility in memory layout options. When the channel fetches the descriptor set, the operation loads this pointer value into the next descriptor pointer register of the DMA channel.

Descriptor Sets

The *Offsets for Descriptor-List Mode Parameters and Descriptors* table shows how to structure a descriptor set in memory. Disperse the placement of the descriptor sets throughout memory, having sets reside in different memory blocks. But, each descriptor of the descriptor set must reside in a contiguous section of memory in the format shown in the table. The values have the same order as the corresponding offset addresses of the memory-mapped registers.

Table 27-10: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current values of the registers between the descriptor set fetch and the start of the DMA operation for the work unit.

Minimum Startup Requirements

At a minimum, write the [DMA_DSCPTR_NXT](#) register prior to write to the [DMA_CFG](#) register, which is the special action required to start the DMA channel.

Descriptor-On-Demand Modes

The [Descriptor-Array Mode](#) and [Descriptor-List Mode](#) each have an on-demand mode of operation.

In on-demand mode, at the end of the work unit, if the DMA channel has not detected an incoming trigger event, the channel repeats the current work unit. If the DMA channel receives an incoming trigger before completion of the work unit, the channel fetches a new descriptor set.

The *Offsets for Descriptor-Array Mode Parameters and Descriptors* and *Offsets for Descriptor-List Mode Parameters and Descriptors* tables illustrate how to structure each descriptor set in memory.

Table 27-11: Offsets for Descriptor-Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

NOTE: For descriptor-array mode, at a minimum, write the [DMA_DSCPTR_CUR](#) register prior to writing to the [DMA_CFG](#) register, which is the special action required to start the DMA channel.

Table 27-12: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

NOTE: For descriptor-list mode, at a minimum, write the [DMA_DSCPTR_NXT](#) register prior to write to the [DMA_CFG](#) register, which is the special action required to start the DMA channel.

Data Transfer Modes

In addition to supporting basic one-dimensional DMA transfers, the DMA channel also supports two-dimensional functionality.

Two-Dimensional DMA

Register-based flow modes and descriptor-based flow modes support two-dimensional data transfers.

In two-dimensional (2D) mode, the X-direction count ([DMA_XCNT](#)), the X-direction modifier ([DMA_XMOD](#)), the Y-direction count ([DMA_YCNT](#)), and the Y-direction modifier ([DMA_YMOD](#)) support arbitrary row and column sizes. Also, the modify values can be negative, allowing implementation of interleaved data streams. The [DMA_XCNT](#) value specifies the row size, and the [DMA_YCNT](#) value specifies the column size; where the [DMA_XCNT](#) value must be 2 or greater.

The DMA start address (`DMA_ADDRSTART`), the X-direction modifier (`DMA_XMOD`), and the Y-direction modifier (`DMA_YMOD`) specifications all are in bytes. The alignment must be a multiple of the DMA transfer word size; configured using the `DMA_CFG.MSIZE` bit. Misalignment results in a DMA channel error.

The `DMA_XMOD` register value is the byte-address increment that the channel applies after each transfer, decrementing the `DMA_XCNT` register. The channel does not apply the `DMA_XCNT` when the inner loop count ends with the `DMA_XCNT_CUR` register decrementing to 0 from 1. Except, the channel does apply the `DMA_XCNT` on the final transfer, when the `DMA_YCNT` register is 1 and the `DMA_XCNT` register decrements from 1 to 0.

The `DMA_YMOD` register value is the byte-address increment that the channel applies after each decrement of the value in `DMA_YCNT_CUR`. However, the channel does not apply the `DMA_YMOD` value to the last item in the array on which the outer loop count (`DMA_YCNT_CUR`) also expires by decrementing from 1 to 0.

After the last transfer completes, `DMA_YCNT_CUR` is 1 and the `DMA_XCNT_CUR` register is 0. The DMA channels current address points to the last items address plus the `DMA_XMOD` register value. If the DMA channel programming selects automatic refresh (such as in autobuffer mode), the channel reloads the `DMA_XCNT_CUR`, `DMA_YCNT_CUR`, and `DMA_ADDR_CUR` for the first data transfer of the next work unit.

Interrupt notification is configurable for end-of-row or end-of-work unit completion.

For example, two-dimensional DMA can be used to extract interleaved data (such as RGB values for a video frame) by modifying both of the `DMA_XMOD` and `DMA_YMOD` values. The *Capturing a Video Data Stream 2D DMA Example* depicts the process of receiving a stream of the R, G, B values from an $N \times M$ frame. The inner loop of the 2D DMA configuration has three values (`DMA_XCNT = 3`) and a stride (`DMA_XMOD`) of $N \times M$, chosen such that successive elements in each row are 1-2-3, 4-5-6 and so forth. The outer loop of the 2D DMA configuration has $N \times M$ values (`DMA_YCNT = N \times M`) and a negative stride (`DMA_YMOD`) of $1 - 2 \times N \times M$ chosen to instruct the DMA controller to jump from element 3 to 4, 6 to 7 and so forth at the end of each inner loop.

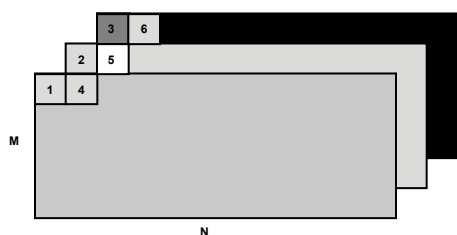


Figure 27-4: Capturing a Video Data Stream 2D DMA Example

DMA Channel Event Control

The DMA channel supports a number of events that provide notification of work unit state, peripheral data request, peripheral interrupt request and completion events, and DMA channel error conditions. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The DMA channel has two interrupt signals for support of a number of events such as work-unit state events, peripheral interrupt request (PIRQ) events, peripheral data request (PDR) events, and DMA channel errors. The

channel reports DMA channel errors on a dedicated interrupt signal. All other interrupt sources share an interrupt signal. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The channel can signal the processor on DMA channel events using status information and optional interrupt requests. Programs can use these events to update the progress of data transfers and to request intervention from the processor core. Configure most DMA channel interrupts using bits in the `DMA_CFG` register. Dedicated bits in the `DMA_STAT` register report the occurrence of various events. Use write-one-to-clear (W1C) operations to clear interrupt requests from the status register.

NOTE: Hardware does not clear the interrupt status bits automatically, even when programs disable then reenables the DMA channel. In this situation, the channel deasserts the interrupt signal, after the program disables the DMA channel. But, the status bit remains set until software either re-enables the DMA channel or clears the status bit.

The DMA channel supports the following categories of events on the interrupt signals:

- Work-unit state events generate interrupts on row or on work unit DMA completion.
- A peripheral uses peripheral interrupt request (PIRQ) events to signal when it has completed the transfer of all data.
- A peripheral uses peripheral data request (PDR) events to request data from a disabled or idle DMA channel.
- Error events signal a failure in the work unit.

ATTENTION: While in an error state, the DMA channel does not generate an interrupt to the processor for a work-unit state event or a PIRQ event, nor does the channel forward a PDR event.

Event Signals

The *Event Signals* table provides descriptions of DMA channel events.

Table 27-13: Event Signals

Signal	Width (bits)	Description
DMA_ERROR	1	Used to signal an error condition in the DMA channel. The source of the error can be determined by reading the <code>DMA_STAT.ERRC</code> bit.
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. Read the corresponding fields in <code>DMA_STAT</code> to determine the source of the event.
DMA_TRIG_OUT	1	Trigger output that gets routed to the TRU and can be configured to provide notification on row or work unit completion.
DMA_TRIG_IN	1	Trigger input from the TRU that can be used to control the start of a work unit.

Work Unit State Events

Completing a row or a work unit generates a work-unit state event. For either of these events to generate an interrupt request, the configuration of the interrupt of the DMA channel must select one of the available work-unit completion modes.

- Current X count reaching 0 for row completion or 1D DMA work unit completion
- Current Y count reaching 0 for work unit completion of 2D DMA

NOTE: For 1D DMA, a DMA channel configuration error results if the configuration generates the interrupt request when the current Y counter reaches 0.

The DMA channel issues the last memory read or write transaction for the row or work unit, then pauses until the return of the read or write acknowledge. After successful acknowledge of the transfer, the DMA channel issues the interrupt request and continues to process the next row or work unit.

Waiting for acknowledgement of the memory access results in a delay. However, programs can read or modify data in the memory without adversely affecting or being affected by the DMA transfer.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

The channel configuration of the synchronization feature also affects interrupt timing. For memory-read operations with synchronization enabled, the channel delays the interrupt request until the completion of the last transfer from the DMA channel FIFO to the peripheral. The synchronization feature does not affect interrupt timing for memory write operations.

Peripheral Interrupt Request Events

For peripheral-transmit operations, a peripheral connected to the DMA channel can use peripheral interrupt request (PIRQ) events to indicate that data has left the channel FIFO and to indicate transfer completion.

In order to support PIRQ interrupts, correctly configure the interrupt of the DMA channel. This configuration disables the generation of interrupt requests based on the work unit state and, instead, results in generating an interrupt request when the DMA channel receives the command from the peripheral.

The channel only generates the interrupt request under the following conditions:

- The configuration enables the DMA channel
- The DMA channel is in the stop state
- The configuration of the DMA channel interrupt selects PIRQ operation

Peripheral Data Request Events

Peripheral data request (PDR) events occur when an interfaced peripheral requests data from the DMA channel and the DMA channel (either disabled or enabled) is in the stop state.

When a peripheral sends a data request command to a disabled DMA channel, the DMA channel generates an interrupt to the System Event Controller (SEC). There is no status information reported about this event in the status register of the DMA channel. Instead, the channel identifies the PDR event from the fact that the DMA channel generated an interrupt while disabled. It is possible to further confirm event status by verifying the status of the peripheral interfaced to the DMA channel.

This operation forwards data requests as interrupts when the DMA channel is in the disabled state. Also, the DMA channel is able to forward PDR events as an interrupt request when the DMA channel is in the stop state after the completion of a work unit. The forwarding of this interrupt when the DMA channel is in the stop state is optional and configured by the program during DMA channel configuration.

DMA Channel Triggers

DMA channel triggers are useful for synchronizing the DMA channel with other events in the system. One usage is to combine channel triggers with each other to create ping-pong buffers. Another usage is to combine the triggers with interrupt requests to notify the processor on reaching a particular milestone that requires service. The channel also can use triggers to enforce a handshake DMA operation in which the trigger acts as a signal for a DMA request.

NOTE: Using the trigger to control the pace of data transfers, such as for handshake DMA, requires that all the data for the entire work unit is ready for transfer.

The DMA channel has a single incoming trigger that can control the pace of the data transfers performed by the DMA channel. The configuration can direct the DMA channel to wait for the incoming trigger before starting the work unit transfer or fetching a descriptor set from memory.

The DMA channel also has a single outgoing trigger signal. This configuration can direct this trigger to signal the end of row or an entire work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

NOTE: The peripheral interrupt (PIRQ) and trigger out are independent functions. PIRQ is asserted when the respective peripheral completes the transaction. Trigger out is asserted when the DMA module completes the transaction.

Issuing Triggers

The DMA channel configuration can direct the channel to generate an outgoing trigger signal at the end of row or the end of a work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until the return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

Waiting For Triggers

Programs can use triggering to control the pace of data transfers performed by the DMA channel. The DMA channel enters a wait state before beginning the next work unit if the configuration enables `DMA_CFG.TWAIT` and either of the following apply:

- The channel receives a trigger since the last time the DMA channel left the wait state.
- The channel receives a trigger since its transition from disable to enable.

In the wait state, the DMA channel also does not perform a descriptor fetch. After receiving a trigger, the DMA channel leaves the wait state and begins the next work unit or fetches the next descriptor if configured for a descriptor-based mode of operation.

If a memory-mapped register write operation programs the channel with stop flow mode enabled (`DMA_CFG.TWAIT` bit) and the channel has not already received a trigger, the DMA channel enters a wait state before performing the data transfer. On receiving the trigger, the DMA channel begins the data transfer portion of the work unit. Once the data transfer is complete, the DMA channel enters the stop state.

If a memory-mapped register write operation programs the DMA channel with the flow mode configured to one of the descriptor-based modes, the DMA channel enters the wait state before performing the descriptor fetch. After completing the descriptor fetch, the DMA channel immediately proceeds to the data transfer, regardless of the value of the `DMA_CFG.TWAIT` bit. If another (next) descriptor fetch follows the descriptor fetch, the DMA channel enters a wait state before fetching the next descriptor.

If the descriptor fetch returns a descriptor with stop flow mode, the `DMA_CFG.TWAIT` value for that descriptor does not affect the DMA as the channel enters the stop state after completing the data transfer. The DMA channel only enters the wait state based on `DMA_CFG.TWAIT` before the next work unit or descriptor fetch.

If the descriptor fetch returns a descriptor configured for autobuffer flow mode, the `DMA_CFG.TWAIT` for that descriptor does not affect the DMA for the first work unit of the autobuffer transfer. After completing the first work unit and not receiving another trigger, the DMA channel enters the wait state before reinitializing its counters and address registers (if not configured for current addressing). The channel performs the next work unit after receiving the trigger.

The incoming trigger can occur when the DMA channel has not entered the wait state. The trigger can occur while the DMA channel is executing a work unit, is performing descriptor fetch, or is in the stop state. The trigger is held internally. After the work unit is complete, the DMA channel skips the wait state and proceeds directly to executing the following work unit. If the `DMA_CFG.TWAIT` bit is not enabled, the DMA channel also skips the wait state. However, the trigger is held internally and is used the next time the configuration enables `DMA_CFG.TWAIT`. This trigger retention allows programs to enable the `DMA_CFG.TWAIT` functionality several work units apart without concern for losing a trigger. The DMA channels trigger-overrun enable functionality can be enabled in all work units to ensure that multiple triggers do not occur between the work units with the `DMA_CFG.TWAIT` bit enabled.

DMA Channel Programming Model

Several synchronization and control methods are available for use in development of software tasks which manage peripheral DMA and memory DMA. Software must accept requests for new DMA transfers from other software tasks, integrate these transfers into existing transfer queues, and reliably notify other tasks when the transfers are complete.

In the processor, it is possible to manage each peripheral DMA and memory DMA stream with a separate task or to manage them together with any other stream. Each DMA channel has independent, orthogonal control registers, resources, and interrupts. So, the selection of the control scheme for one channel does not affect the choice of control scheme on other channels. For example, one peripheral can use a linked-descriptor-list, interrupt-driven scheme while another peripheral can simultaneously use a demand-driven, buffer-at-a-time scheme synchronized by polling DMA events.

The topics that follow describe the steps required to configure the DMA channel for the various modes in addition to the programming concepts required for software synchronization.

Mode Configuration

Use the step-by-step directions that follow to set up the DMA channel for operating modes.

Register-Based Linear-Buffer Stop Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and to send it to the peripheral for transmission. Upon DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

Assume that the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

1. Write the `DMA_ADDRSTART` register.

ADDITIONAL INFORMATION: Software can use the address to calculate the most optimum possible `DMA_CFG.MSIZE`.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE`, and the calculation also must consider the start address alignment.

3. Write the `DMA_XCNT` register based on the calculated `DMA_CFG.MSIZE`.

ADDITIONAL INFORMATION: The `DMA_XCNT` value is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register.

ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. Always specify this register as a number of bytes.

5. Write the `DMA_CFG` register with `DMA_CFG.EN` configured to enable the DMA channel.

ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` bit for STOP mode. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus. From performance perspective, it is recommended to use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `DMA_CFG.PSIZE` value used.

- The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer is transferred. The DMA channel enters the IDLE state upon completion of the work unit.

Register-Based Autobuffer Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and send it to the peripheral for transmission. The transmission of the buffer repeats endlessly. Upon DMA completion, the DMA channel restarts the DMA operation, creating an endless circular buffer transfer.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

1. Write the `DMA_ADDRSTART` register.

ADDITIONAL INFORMATION: Use the address to calculate the optimum possible `DMA_CFG.MSIZE`.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE`, and the calculation must consider the start address alignment.

3. Write the `DMA_XCNT` register based on calculated `DMA_CFG.MSIZE`.

ADDITIONAL INFORMATION: The `DMA_XCNT` register value is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register.

ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. Always specify this register as a number of bytes.

5. Write the `DMA_CFG` register with the `DMA_CFG.EN` bit configured to enable the DMA channel.

ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` bit for autobuffer mode. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bit to a value no larger than the supported bus width of the peripheral DMA bus. From performance perspective, it is recommended to use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `DMA_CFG.PSIZE` value used.

- The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer transfers until the DMA channel is disabled.

Descriptor-Array Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the array, and
- Send the data to the peripheral for transmission.

Descriptor sets are read from an array in memory to configure the individual work units.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel. Assume that the array of descriptors is to be initialized with the last descriptor set configured for STOP flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the array in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_DSCPTR_CUR` register with the address of the array in which the descriptor sets are stored.

ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the `DMA_CFG` register with the `DMA_CFG.EN` bit configured to enable the DMA channel.

ADDITIONAL INFORMATION: Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus. From performance perspective, it is recommended to use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `DMA_CFG.PSIZE` value used. Set the `DMA_CFG.FLOW` bit for descriptor-array mode. Configure the `DMA_CFG.NDSIZE` bits to describe the number of descriptor elements contained within the first descriptor set. Configure the `DMA_CFG.WNR` bit for memory read operation.

- The descriptor set that is fetched controls the `DMA_CFG.SYNC` configuration and the interrupt or trigger configurations.

The first descriptor set is fetched from memory location provided by the `DMA_DSCPTR_CUR` register and loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor array. The DMA channel enters the IDLE state on completion of the final work unit that was configured for STOP flow mode.

Descriptor-List Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the list, and
- Send it to the peripheral for transmission.

The DMA controller reads the descriptor sets from a list of descriptors. With the list, each descriptor set has a descriptor that points to the next descriptor set location in memory.

Assume the peripheral must be in a state where it is ready to transmit data received from the DMA channel. Assume that the list of descriptors must be initialized with the last descriptor set in the list configured for Stop flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the list in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_DSCPTR_NXT` register with the address of the first descriptor in the list to be processed.

ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the `DMA_CFG` register with the `DMA_CFG.EN` configured to enable the DMA channel.

ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` for descriptor-list mode. Configure the `DMA_CFG.NDSIZE` bit to describe the number of descriptor elements contained within the first descriptor set. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bit to a value no larger than the supported bus width of the peripheral DMA bus. From performance perspective, it is recommended to use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `DMA_CFG.PSIZE` value used.

- The descriptor set that is fetched controls the `DMA_CFG.SYNC` configuration and controls the interrupt or trigger configurations.

The first descriptor set is fetched from the memory location provided by `DMA_DSCPTR_NXT` and is loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor list. The DMA channel enters the idle state when the final work unit that was configured for stop-flow mode is complete.

Register-Based Memory-to-Memory Transfer in Stop Flow Mode

This procedure configures a memory DMA channel pair in stop flow mode. One DMA channel is configured for memory read operations, while the other DMA channel is configured for memory write.

The task involves writing to a number of DMA channels on two DMA channels that create a memory DMA pair. On DMA completion, the DMA channel enters the idle state, until either the DMA channel is disabled or is reconfigured for a new transfer.

1. Write the `DMA_ADDRSTART` register of the source DMA channel.

ADDITIONAL INFORMATION: The address can be used to calculate the optimum `DMA_CFG.MSIZE` possible.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE` and the start address alignment must also be considered.

3. Write the `DMA_XCNT` register of the source DMA channel based on calculated `DMA_CFG.MSIZE`.

ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register of the source DMA channel.

ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

5. Write the `DMA_ADDRSTART` register of the destination DMA channel.

ADDITIONAL INFORMATION: The address can be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

6. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE` and the start address alignment must also be considered.

7. Write the `DMA_XCNT` register of the destination DMA channel based on the calculated `DMA_CFG.MSIZE`.

ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

8. Write the `DMA_XMOD` register of the destination DMA channel.

ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

9. Write the `DMA_CFG` register of the source DMA channel with `DMA_CFG.EN` configured to enable the DMA channel.

ADDITIONAL INFORMATION: The `DMA_CFG.FLOW` bit field must be configured for stop mode. The `DMA_CFG.WNR` bit must be cleared for memory read operation. The `DMA_CFG.PSIZE` bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus. From performance perspective, it is recommended to use the largest possible `DMA_CFG.PSIZE` value (for example, equal to the supported peripheral bus width (`DMA_STAT.PBWID`)). However, ensure that the number of bytes in the work unit is a multiple of the `DMA_CFG.PSIZE` value used.

- The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step, depending on requirements. The interrupts and triggers are enabled within the destination DMA channel configuration.

The memory read DMA transfer begins.

10. Write the `DMA_CFG` register of the destination DMA channel with `DMA_CFG.EN` configured to enable the DMA channel.

ADDITIONAL INFORMATION: The `DMA_CFG.FLOW` bit field must be configured for stop mode. The `DMA_CFG.WNR` bit must be set for memory write operation. The `DMA_CFG.PSIZE` bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus. This value must also match the value written for the source DMA channel configuration.

- Interrupts and triggers also can be configured at this step depending on requirements.

The memory write DMA transfer begins.

Both memory DMA channels are now running and the data is transferred from the source address to the destination address. The DMA channel enters the IDLE state upon completion of the work unit.

Programming Concepts

Using the features, operating modes, and event control for the DMA channel to their greatest potential requires an understanding of some DMA channel-related concepts.

Synchronization of Software and DMA

A critical element of software DMA management is the synchronization of DMA work unit completion with software. This synchronization can be achieved using DMA channel interrupt request and trigger events and using a poll of the status bits of these events within the DMA channel registers, or combining these techniques. Processor polling of DMA address/count/status for completion is not a recommended programming practice. The requirements and limitations of processor polling place significant responsibility onto the code developer to be deeply aware of the underlying hardware. The interrupt requests and triggers are designed for efficient code development and reuse.

Interrupt and Trigger Event-Based Synchronization

Interrupt and trigger based synchronization methods must avoid overrun. An overrun occurs when some events fail to invoke the event handler of a DMA channel for every event due to excessive latency in processing of events. The system design must ensure to either:

- Schedule only one event per channel (for example, at the end of a descriptor list), or
- Space the generated events sufficiently far apart in time that system processing budgets can guarantee service of every event.

The DMA channel issues status information through an interrupt request or trigger event or changes event status bits in the `DMA_STAT` register. This status guarantees that the last memory operation of the work unit is complete. For memory read DMA transactions, this status means that the FIFO of the DMA channel safely receives the final memory read data. For DMA transactions writing to memory, this status indicates that the DMA channel received an acknowledge of completion of the last write transfer of the work unit.

Register Polling Based Synchronization

Do not poll the DMA channel registers (`DMA_ADDR_CUR`, `DMA_DSCPTR_CUR`, `DMA_XCNT_CUR`, or `DMA_YCNT_CUR`) as a method of precisely synchronizing DMA with data processing. This approach is inaccurate due to the operation of the DMA channel FIFOs and DMA or memory pipelining. The current address, pointer, and count registers change several cycles in advance of the completion of the corresponding memory operation. This timing is measurable from the time at which the results of the operation are first visible to the core by memory read or write instructions.

For example, in a DMA channel memory write operation to external memory, assume DMA channel *A* initiates a DMA channel write operation. For memories with access latency, this operation requires many system-clock cycles. Meanwhile, DMA channel *B* (which does not in itself incur latency) initiates a transfer, which stalls behind the slow operation of channel *A*. Software monitoring channel *B* could not safely conclude whether the memory location pointed to by the `DMA_ADDR_CUR` of channel *B*. Also, the software cannot conclude whether the register has been written based solely on the contents of this register.

Polling of the current address, pointer, and count registers can permit loose synchronization of DMA with software. But, the software must allow for the lengths of the DMA or memory pipeline. Also, software must consider the length of the DMA FIFO for a particular peripheral. If the FIFOs are filled with incomplete work, the DMA channel does not advance current address, pointer, or count registers. The incomplete work includes reads that have been started but have not yet finished.

Additionally, software must consider the length of the pipelines to the destination memory. If the DMA FIFO length and channel memory-pipeline length are added, software can estimate the maximum number of incomplete memory operations in progress.

NOTE: The estimate would be a maximum, as the DMA or memory pipeline can include traffic from other DMA channels.

Descriptor Queues

A system designer may want to write a DMA manager facility which accepts DMA requests from other software. The DMA manager software does not know in advance when new work requests are received or what these requests contain. The software could manage these transfers using a circular linked list of DMA descriptors. In such a list, each descriptor sets the `DMA_DSCPTR_NXT` descriptor, which points to the next descriptor set. And, the last descriptor set in the list points to the first descriptor set.

The code that writes into this descriptor list could use the circular addressing modes of the processor. This approach does not need to use comparison and conditional instructions to manage the circular structure. In this case, the `DMA_DSCPTR_NXT` descriptor of each descriptor set can be written once at startup, and skipped over as new contents are written for each descriptor.

The recommended method for synchronization of a descriptor queue is to use an interrupt or trigger. The descriptor queue is structured, such that (at least) the final valid descriptor set is always programmed to generate an interrupt or trigger event upon completion. More detail is provided in the following sections.

- [Queues Using Event Generation for Every Descriptor Set](#)
- [Queues Using Minimal Events](#)

Queues Using Event Generation for Every Descriptor Set

In this system, the DMA manager software synchronizes with the DMA channel by enabling an interrupt request or trigger on every descriptor set. Only use this method if the system design can guarantee that each work unit completion event is serviced separately (no interrupt or trigger overrun).

To maintain synchronization of the descriptor set queue, the non-interrupt software maintains a count of descriptor sets added to the queue. The event handler (either interrupt or trigger) maintains a count of completed descriptor sets removed from the queue. The counts are equal only when the DMA channel is paused after having processed all the descriptor sets.

When each new work unit event is received, the DMA manager software initializes a new descriptor set, taking care to set the flow to stop mode. Next, the software compares the descriptor set counts to determine whether the DMA channel is running. If the DMA channel is paused (counts equal), the software increments its count. Then, the software starts the DMA channel by writing the `DMA_CFG` of the new descriptor set.

If the counts are unequal, the software instead modifies the `DMA_CFG` of the next-to-last descriptor set, such that it now describes the newly queued descriptor set. This operation does not disrupt the DMA channel provided the rest of the descriptors of the set are initialized in advance. It is necessary to synchronize the software to the DMA to determine whether the DMA channel read the new or the old `DMA_CFG` value.

The event handler performs the synchronization operation. When an event is detected, the handler reads the `DMA_STAT` register of the DMA channel. If the `DMA_STAT.RUN` bit indicates that the DMA channel is running, the channel has moved on to processing another descriptor. The event handler can increment its count and exit. If the `DMA_STAT.RUN` bit indicates that the channel is not running, the channel is paused because either:

- There are no more descriptor sets to process, or
- The last descriptor set was queued too late

Where *too late* means that the modification of the `DMA_CFG` of the next-to-last descriptor set occurred *after* that descriptor was read into the DMA channel. In this case, the event handler does the following:

- Writes the `DMA_CFG` value appropriate for the last descriptor set to `DMA_CFG` register of the DMA channel,
- Increments the completed descriptor count, and
- Exits

If the event latencies of the system are large enough to cause any of the events to be dropped, this system can fail. An event handler capable of safely synchronizing multiple descriptor set interrupt requests is complex, performing several MMR accesses to ensure robust operation. In such a system environment, a minimal event synchronization method is preferred.

Queues Using Minimal Events

In this system, only one DMA interrupt request or trigger event is generated in the queue at any time. The DMA event handler for this system can also be extremely short. Here, the descriptor queue is organized into an *active* and a *waiting* portion, where events are enabled only on the last descriptor set in each portion.

When each new DMA request is processed, the software fills in the content of a new descriptor set and adds it to the waiting portion of the queue. The `DMA_CFG` descriptor of the descriptor set must have the flow set to stop mode. If more than one request is received before the DMA queue completion event occurs, the non-interrupt code queues later descriptor sets. It forms a waiting portion of the queue separate from the active portion of the queue that the DMA channel is processing. In other words, all but the last active descriptor sets contain flow values for a descriptor-based mode and have no event enable set.

The last active descriptor set has the stop flow mode and an event generation enabled. Also, all but the last waiting descriptor sets are configured for descriptor-based flow modes with no event generation. Only the last waiting descriptor set is configured for stop flow mode and event generation enabled. This configuration ensures that the DMA channel can automatically process the whole active queue before then issuing one event. Also, this arrangement makes it easy to start the waiting queue within the event handler by a single `DMA_CFG` register write.

After queuing a new waiting descriptor, the non-interrupt software leaves a message for its interrupt handler in a memory mailbox location. The location contains the desired `DMA_CFG` value for starting the first waiting descriptor set in the waiting queue (or 0, indicating no waiting descriptors).

The software must not modify the contents of the active descriptor set queue directly once processing by the DMA channel has started, unless careful synchronization measures are taken. In the most straightforward implementation of a descriptor set queue, the DMA manager software never modifies descriptors on the active queue. Instead, the DMA manager waits until the DMA queue completion event indicates that the processing of the entire active queue is complete.

When a DMA queue completion event is received, the event handler reads the mailbox from the non-interrupt software and writes the value to the `DMA_CFG` register of the DMA channel. This write to a register restarts the queue, effectively transforming the waiting queue to an active queue. The event handler then passes a message back to the non-interrupt software indicating the location of the last descriptor set accepted into the active queue.

However, the event handler can read its mailbox and find a `DMA_CFG` value of zero, indicating there is no more work to perform. It then passes an appropriate message back to the non-interrupt software indicating that the queue has stopped.

The non-interrupt software which accepts new DMA work unit requests must synchronize the activation of a new work unit with the interrupt handler. If the queue has stopped (the mailbox from the event handler is zero), the non-interrupt software must start the queue. (The queue starts by writing the first descriptor sets `DMA_CFG` value to the `DMA_CFG` register of the channel). If the queue is not stopped, the non-interrupt software must not write the `DMA_CFG` register. (This write causes a DMA error). Instead, it must queue the descriptor onto the waiting queue and update its mailbox directed to the event handler.

ADSP-2156x DMA Register Descriptions

The Direct Memory Access module (DMA) contains the following registers.

Table 27-14: ADSP-2156x DMA Register List

Name	Description
<code>DMA_ADDRSTART</code>	Start Address of Current Buffer Register
<code>DMA_ADDR_CUR</code>	Current Address Register
<code>DMA_BWLCNT</code>	Bandwidth Limit Count Register
<code>DMA_BWLCNT_CUR</code>	Bandwidth Limit Count Current Register
<code>DMA_BWMCNT</code>	Bandwidth Monitor Count Register
<code>DMA_BWMCNT_CUR</code>	Bandwidth Monitor Count Current Register
<code>DMA_CFG</code>	Configuration Register
<code>DMA_DSCPTR_CUR</code>	Current Descriptor Pointer Register
<code>DMA_DSCPTR_NXT</code>	Pointer to Next Initial Descriptor Register

Table 27-14: ADSP-2156x DMA Register List (Continued)

Name	Description
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

Start Address of Current Buffer Register

The `DMA_ADDRSTART` register contains the start address of the work unit currently targeted for DMA. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

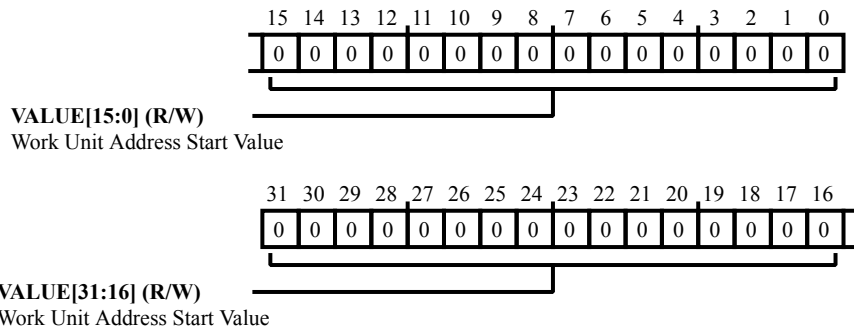


Figure 27-5: `DMA_ADDRSTART` Register Diagram

Table 27-15: `DMA_ADDRSTART` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Address Start Value. The <code>DMA_ADDRSTART.VALUE</code> bit field contains the start address of the work unit currently targeted for DMA.

Current Address Register

The `DMA_ADDR_CUR` register contains the present memory transfer address for a given work unit. At the start of a work unit, the `DMA_ADDR_CUR` register is loaded from the `DMA_ADDRSTART` register, and the `DMA_ADDR_CUR` register is incremented as each transfer occurs. The `DMA_ADDR_CUR` register is read/write prior to enabling the channel, but is read-only after enabling the channel.

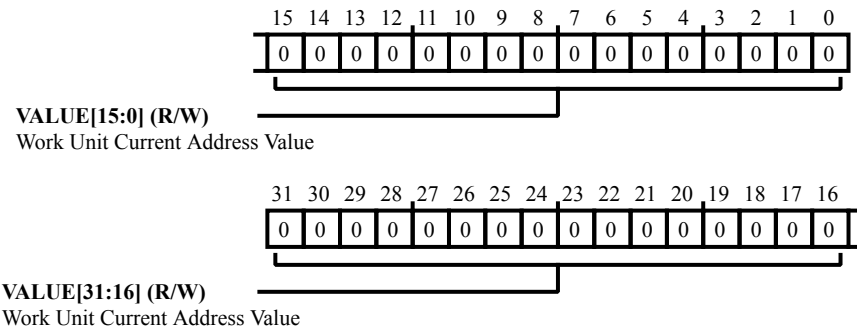


Figure 27-6: `DMA_ADDR_CUR` Register Diagram

Table 27-16: `DMA_ADDR_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Current Address Value. The <code>DMA_ADDR_CUR.VALUE</code> bit field contains the present memory transfer address for a given work unit.

Bandwidth Limit Count Register

The `DMA_BWLCNT` register contains a count that determines how often the DMA issues memory transactions. The DMA loads the value from the `DMA_BWLCNT` register into the `DMA_BWLCNT_CUR` register and decrements the current value each SCLK cycle. When `DMA_BWLCNT_CUR` reaches 0x0000, the next request is issued, and the DMA reloads `DMA_BWLCNT_CUR`. This bandwidth limit functionality is not applied to descriptor fetch requests. Programming 0x0000 allows the DMA to request as often as possible. 0xFFFF is a special case and causes requests to stop.

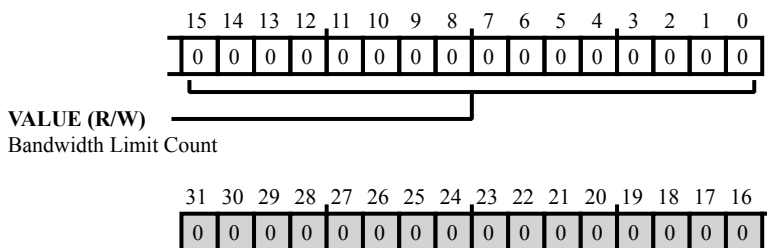


Figure 27-7: `DMA_BWLCNT` Register Diagram

Table 27-17: `DMA_BWLCNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Bandwidth Limit Count. The <code>DMA_BWLCNT.VALUE</code> bit field contains a count that determines how often the DMA issues memory transactions.

Bandwidth Limit Count Current Register

The `DMA_BWLCNT_CUR` register contains the number of SCLK count cycles remaining before the next request is issued.

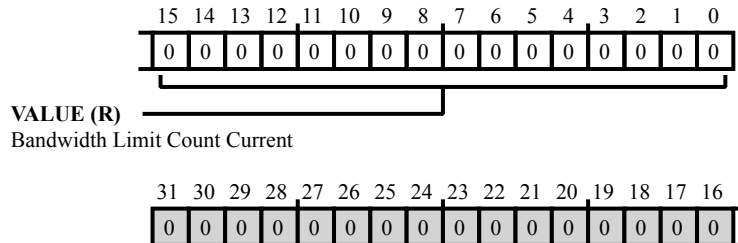


Figure 27-8: DMA_BWLCNT_CUR Register Diagram

Table 27-18: DMA_BWLCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Bandwidth Limit Count Current. The <code>DMA_BWLCNT_CUR.VALUE</code> bit field contains the number of SCLK count cycles remaining before the next request is issued.

Bandwidth Monitor Count Register

The `DMA_BWMCNT` register contains the maximum number of SCLK cycles allowed for a work unit to complete. Each time the `DMA_CFG` register is written (MMR access only), a work unit ends or an autobuffer wraps. The DMA loads the value in this register into the `DMA_BWMCNT_CUR` register.

The DMA decrements `DMA_BWMCNT_CUR` every SCLK a work unit is active. If the `DMA_BWMCNT_CUR` register reaches 0x0000_0000, the `DMA_STAT.IRQERR` bit is set, and the `DMA_STAT.ERRC` bit field is set to 0x6. The `DMA_BWMCNT_CUR` remains at 0x0000_0000 until it is reloaded when the work unit completes.

Unlike other errors, a bandwidth monitor error does not stop work unit processing. Programming 0x0000_0000 disables bandwidth monitor functionality.

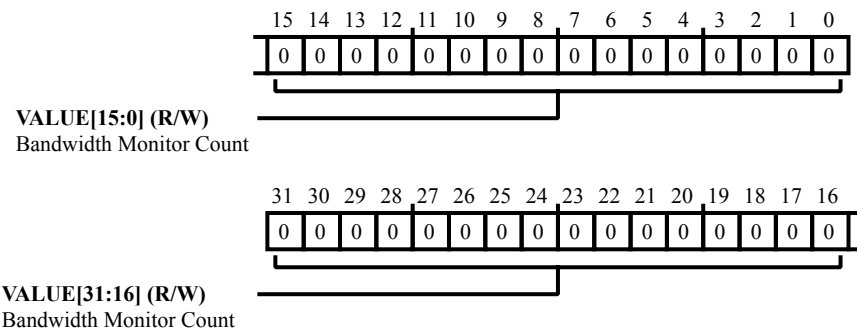


Figure 27-9: `DMA_BWMCNT` Register Diagram

Table 27-19: `DMA_BWMCNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Bandwidth Monitor Count. The <code>DMA_BWMCNT.VALUE</code> bit field contains the maximum number of SCLK cycles allowed for a work unit to complete.

Bandwidth Monitor Count Current Register

The `DMA_BWMCNT_CUR` register contains the number of cycles remaining for the current descriptor to complete.

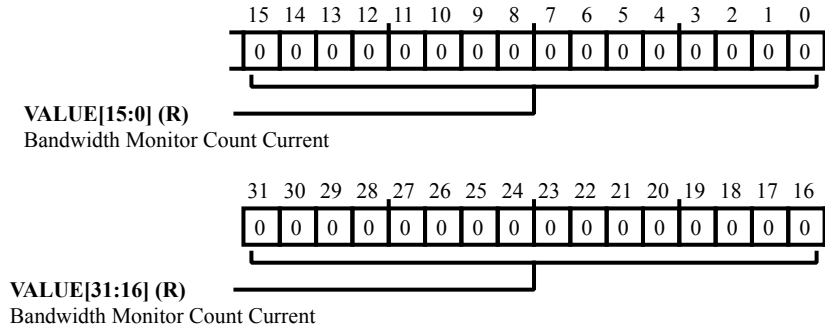


Figure 27-10: `DMA_BWMCNT_CUR` Register Diagram

Table 27-20: `DMA_BWMCNT_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bandwidth Monitor Count Current. The <code>DMA_BWMCNT_CUR.VALUE</code> bit field contains the number of cycles remaining for the current descriptor to complete.

Configuration Register

The `DMA_CFG` register sets up DMA parameters and operation modes. Writing to the `DMA_CFG` register while a DMA process is already running causes a DMA error (except when clearing the `DMA_CFG.EN` bit).

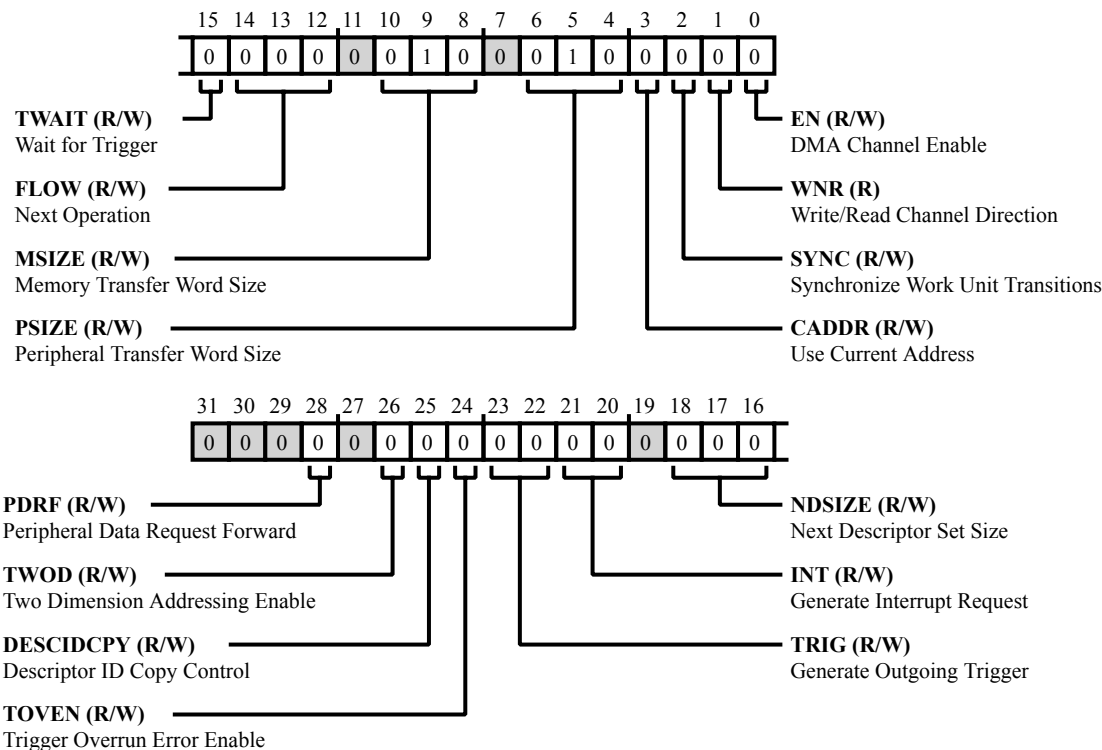


Figure 27-11: `DMA_CFG` Register Diagram

Table 27-21: `DMA_CFG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	PDRF	Peripheral Data Request Forward. The <code>DMA_CFG.PDRF</code> bit defines how the DMA handles data requests from the peripheral while in idle state after a stop mode or memory read work unit. If set, the DMA forwards the peripheral data request as an interrupt. This bit applies only to the <code>DMA_CFG.FLOW</code> bits configured for stop and <code>DMA_CFG.WNR</code> bits configured for memory read.
		0 Peripheral Data Request Not Forwarded
		1 Peripheral Data Request Forwarded

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	TWOD	Two Dimension Addressing Enable. The DMA_CFG.TWOD bit selects whether the DMA addressing involves only DMA_XCNT and DMA_XMOD (one-dimensional DMA) or also involves DMA_YCNT and DMA_YMOD (two-dimensional DMA).
		0 One-Dimensional Addressing
		1 Two-Dimensional Addressing
25 (R/W)	DESCIDCPY	Descriptor ID Copy Control. The DMA_CFG.DESCIDCPY bit specifies when to copy the initial descriptor pointer to the DMA_DSCPTR_PRV register. A bus write to the DMA_CFG register to clear the DMA_CFG.EN bit causes the DMA to immediately use the new value of the DMA_CFG.DESCIDCPY bit. To preserve consistency (if required by application), match the new value of this bit to the previous value.
		0 Never Copy
		1 Copy on Work Unit Complete
24 (R/W)	TOVEN	Trigger Overrun Error Enable. A trigger overrun occurs if more than one trigger was received before the DMA reached the trigger wait state. If DMA_CFG.TOVEN is set, a trigger overrun causes the DMA to flag an error. In cases where a trigger overrun is not a problem, clearing DMA_CFG.TOVEN prevents the overrun from causing an error and halting the DMA. The DMA_CFG.TOVEN operates independently of the DMA_CFG.TWAIT bit selection.
		0 Ignore Trigger Overrun
		1 Error on Trigger Overrun

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration								
23:22 (R/W)	TRIG	<p>Generate Outgoing Trigger.</p> <p>The DMA_CFG.TRIG selects whether the DMA issues an outgoing trigger, based on the work unit counter values. In one-dimensional mode, the only options are to trigger at the end of the whole work unit (trigger when DMA_XCNT_CUR reaches 0) or not to trigger at all. If in one-dimensional addressing mode, programming the DMA_CFG.TRIG bit field to trigger when DMA_YCNT_CUR reaches 0 (or to reserved) causes the DMA to flag a configuration error.</p> <p>In two-dimensional addressing mode, the trigger options are: at the end of each row of the inner loop (when DMA_XCNT_CUR reaches 0), only after completing the whole work unit (when DMA_YCNT_CUR reaches 0), or no trigger. If in two-dimensional mode and set to trigger when DMA_XCNT_CUR reaches 0, the DMA also issues a trigger at the end of the work unit. If in two-dimensional addressing mode, programming DMA_CFG.TRIG to reserved causes the DMA to flag a configuration error.</p> <p>If DMA_CFG.TRIG is non-zero and the peripheral issues a finish command, the DMA issues a trigger after the finish procedure is complete.</p> <p>For more information about trigger generation timing, see the trigger section of the DMA functional description.</p> <table border="1"> <tbody> <tr> <td>0</td> <td>Never Assert Trigger</td> </tr> <tr> <td>1</td> <td>Trigger When XCNTCUR Reaches 0</td> </tr> <tr> <td>2</td> <td>Trigger When YCNTCUR Reaches 0</td> </tr> <tr> <td>3</td> <td>Reserved</td> </tr> </tbody> </table>	0	Never Assert Trigger	1	Trigger When XCNTCUR Reaches 0	2	Trigger When YCNTCUR Reaches 0	3	Reserved
0	Never Assert Trigger									
1	Trigger When XCNTCUR Reaches 0									
2	Trigger When YCNTCUR Reaches 0									
3	Reserved									

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21:20 (R/W)	INT	Generate Interrupt Request. The DMA_CFG . INT bit field selects whether an interrupt request goes to the core based on work unit status or a peripheral interrupt request. For one-dimensional mode, setting the DMA_CFG . INT bits to generate an interrupt request when the DMA_YCNT_CUR register reaches 0 causes the DMA to flag a configuration error. The peripheral interrupt setting lets the DMA generate the last grant indication and to accept and or forward the peripheral interrupt command. The peripheral interrupt selection applies only to the DMA_CFG . FLOW bits set for stop and the DMA_CFG . WNR bits set for memory read. If the DMA_CFG . INT is set for interrupt request on count completion (DMA_XCNT_CUR or DMA_YCNT_CUR reach 0) and the peripheral issues a finish command, the DMA issues an interrupt request after the finish procedure is complete. For more information, see the sections on interrupt generation and peripheral control in the DMA functional description.
		0 Never Assert Interrupt
		1 Interrupt When X Count Expires
		2 Interrupt When Y Count Expires
		3 Peripheral Interrupt request
18:16 (R/W)	NDSIZE	Next Descriptor Set Size. The DMA_CFG . NDSIZE bit field specifies the number of descriptor elements in memory to load during the next descriptor fetch. The DMA loads the descriptors in a specific order. The descriptor set contains the next descriptor pointer when it is a descriptor list. The descriptor set does not contain the next descriptor pointer when it is a descriptor array.
		0 Fetch One Descriptor Element
		1 Fetch Two Descriptor Elements
		2 Fetch Three Descriptor Elements
		3 Fetch Four Descriptor Elements
		4 Fetch Five Descriptor Elements
		5 Fetch Six Descriptor Elements
		6 Fetch Seven Descriptor Elements
7 Reserved		

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	TWAIT	Wait for Trigger. The DMA_CFG.TWAIT bit controls whether the DMA waits for an incoming trigger from another channel or user. If the DMA_CFG.TWAIT bit is set, the DMA enters the wait state before starting the next work unit, including descriptor fetch when in descriptor mode. Do not use the wait for trigger control for descriptor-on-demand mode which causes an error. For more information, see the trigger section of the DMA functional description.
		0 Begin Work Unit Automatically (No Wait)
		1 Wait for Trigger (Halt before Work Unit)
14:12 (R/W)	FLOW	Next Operation. The DMA_CFG.FLOW bit field selects the descriptor handling options.
		0 STOP. See the Stop Flow Mode section.
		1 AUTO. See the Autobuffer Flow Mode section.
		2 Reserved
		3 Reserved
		4 DSCL. See the Descriptor List Mode section.
		5 DSCA. See the Descriptor Array Mode section.
		6 Descriptor On-Demand List. See the Descriptor List Mode section.
7 Descriptor On Demand Array. See the Descriptor On Demand section.		
10:8 (R/W)	MSIZE	Memory Transfer Word Size. The DMA_CFG.MSIZE bits select memory transfer sizes of 8-, 16-, 32-, 64-, 128-, or 256-bit words. The transfer start address (DMA_ADDRSTART) and transfer increment values (DMA_XMOD, and, if needed, DMA_YMOD) must be a multiple of the memory transfer unit size.
		2 4 Bytes
		3 8 Bytes
		4 16 Bytes
		5 32 Bytes

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:4 (R/W)	PSIZE	Peripheral Transfer Word Size. The DMA_CFG.PSIZE bits select peripheral transfer sizes as 8, 16, 32, or 64 bits wide. Each request and grant results in a single peripheral access. There are no bursts on the peripheral bus, so the DMA_CFG.PSIZE selection must be less than or equal to the width of the bus. If the selection is greater than the bus width, a configuration error occurs. The peripheral bus of the processor is dedicated to DMA and peripheral accesses.
		2 4 Bytes
		3 8 Bytes
3 (R/W)	CADDR	Use Current Address. When the DMA_CFG.CADDR bit is cleared, the DMA loads the DMA_ADDRSTART register on the first access of the work unit. When the DMA_CFG.CADDR bit is set, the DMA uses the DMA_ADDR_CUR register value for the starting address for the work unit and writes the same value to the DMA_ADDRSTART register. This operation permits continuation of a previous work unit. When DMA uses this mode, the fetched start address value (as part of the descriptor set at the end of a descriptor list or array) is ignored.
		0 Load Starting Address
		1 Use Current Address

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	SYNC	<p>Synchronize Work Unit Transitions.</p> <p>Setting the <code>DMA_CFG.SYNC</code> bit clears the DMA FIFO and pointers before starting the first work unit of a work unit chain.</p> <p>When the transfer direction is memory read/transmit (<code>DMA_CFG.WNR = 0</code>), the DMA waits until all data transmits to a peripheral before moving on to the next work unit, clearing the FIFO and pointers.</p> <p>When the transfer direction is memory write/receive (<code>DMA_CFG.WNR = 1</code>), the DMA ignores the <code>DMA_CFG.SYNC</code> bit value after processing the first work unit of a work unit chain. As a channel can receive data when turned on but idle, data from the peripheral can still be in the FIFO even though the channel was not programmed. When the <code>DMA_CFG.SYNC</code> bit field is set at the beginning of a work unit chain (during the first work unit), the DMA clears the FIFO, erasing the data put into the FIFO while the channel was idle.</p> <p>Syncing lets programs change the <code>DMA_CFG.PSIZE</code> between individual work units and (in some cases) work unit chains. The sync resets the pointers in the FIFO, preventing misaligned FIFO access.</p> <p>Programs can change the <code>DMA_CFG.MSIZE</code> field between consecutive work units, independent of the <code>DMA_CFG.SYNC</code> bit setting.</p> <p>Syncing also permits changes to transfer direction. Because the data in the FIFO is eliminated, the data that went into the FIFO from one direction (transmit or receive) is not sent back in the other direction after the direction change.</p>
		0 No Synchronization
		1 Synchronize Channel
1 (R/NW)	WNR	<p>Write/Read Channel Direction.</p> <p>The <code>DMA_CFG.WNR</code> selects receive (write to memory) or transmit (read from memory) channel direction.</p>
		0 Transmit (Read from memory)
		1 Receive (Write to memory)

Table 27-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	<p>DMA Channel Enable.</p> <p>The <code>DMA_CFG.EN</code> bit enables the selected DMA channel.</p> <p>When a peripheral DMA channel is enabled, data requests from the peripheral denote DMA requests. When a channel is disabled, the DMA unit ignores the peripheral data request and passes it directly to the system event controller.</p> <p>To avoid unexpected results, enable the DMA channel before enabling the peripheral, and disable the peripheral before disabling the DMA channel.</p> <p>A transition of the <code>DMA_CFG.EN</code> bit from 0 to 1 creates a hard reset of all internal counters and states, including the <code>DMA_STAT</code> register. (All other register values remain unaffected.) A transition from 1 to 0 maintains all counters and registers for reading and analysis.</p> <p>Note that if a descriptor loads when this bit is cleared (see the <code>DMA_CFG.FLOW</code> field), the DMA transitions to the off or idle state after the descriptor load is complete.</p>	
		0	Disable
		1	Enable

Current Descriptor Pointer Register

The `DMA_DSCPTR_CUR` register contains the memory address for the next descriptor to be loaded. The `DMA_DSCPTR_CUR` register is read/write prior to enabling the channel, but is read-only after enabling the channel. For `DMA_CFG.FLOW` mode settings that involve descriptor fetches, this register is used to read descriptors into appropriate MMRs before a work unit begins. For descriptor list mode, the `DMA_DSCPTR_CUR` register is initialized from the `DMA_DSCPTR_NXT` register before fetching each descriptor set. Then, the address in the `DMA_DSCPTR_CUR` register increments as each descriptor is read.

When the entire descriptor set has been read, the `DMA_DSCPTR_CUR` register contains this value:

$$\text{DMA_DSCPTR_CUR} = \text{Descriptor Start Address} + \text{Descriptor Size} (\# \text{ of elements})$$

For descriptor array mode, the `DMA_DSCPTR_CUR` register, and not the `DMA_DSCPTR_NXT` register, must be programmed by a MMR access before starting DMA operation.

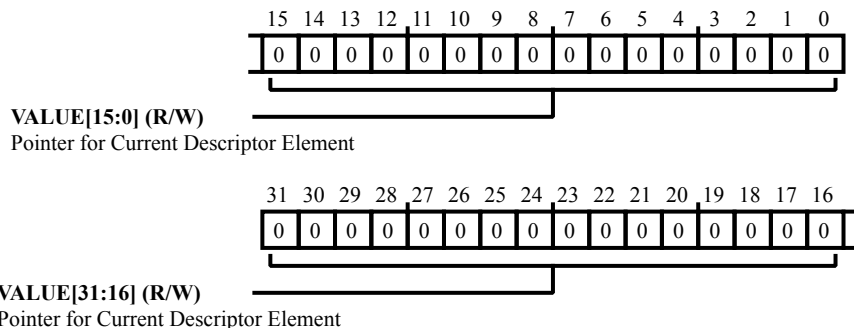


Figure 27-12: `DMA_DSCPTR_CUR` Register Diagram

Table 27-22: `DMA_DSCPTR_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer for Current Descriptor Element. The <code>DMA_DSCPTR_CUR.VALUE</code> bit field contains the memory address for the next descriptor to be loaded.

Pointer to Next Initial Descriptor Register

The `DMA_DSCPTR_NXT` register specifies the start location of the next descriptor set, which begins when the DMA activity specified by the current descriptor set completes. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

The `DMA_DSCPTR_NXT` register is only used in descriptor list mode. At the start of a descriptor fetch in this mode, the `DMA_DSCPTR_NXT` register is copied into the `DMA_DSCPTR_CUR` register. During descriptor fetch, the DMA increments the `DMA_DSCPTR_CUR` register value after reading each element of the descriptor set.

In descriptor list mode, the `DMA_DSCPTR_NXT` register (not the `DMA_DSCPTR_CUR` register) must be programmed directly through MMR access, before the DMA operation is started. In descriptor array mode, the DMA disregards the `DMA_DSCPTR_NXT` register and uses the `DMA_DSCPTR_CUR` register to control descriptor fetch.

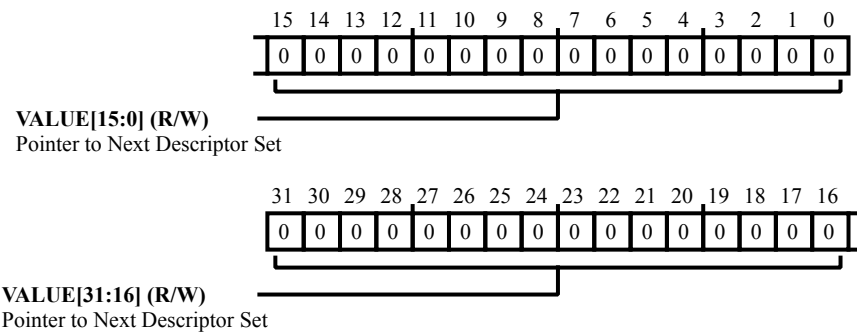


Figure 27-13: `DMA_DSCPTR_NXT` Register Diagram

Table 27-23: `DMA_DSCPTR_NXT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer to Next Descriptor Set. The <code>DMA_DSCPTR_NXT.VALUE</code> bit field specifies the start location of the next descriptor set.

Previous Initial Descriptor Pointer Register

The `DMA_DSCPTR_PRV` register contains the initial descriptor pointer for the previous work unit. If `DMA_CFG.DESCIDCPY` is set, the DMA copies the initial descriptor pointer to `DMA_DSCPTR_PRV` after the work unit completes. Otherwise, the value is not updated.

To indicate an overrun, bit 0 of the `DMA_DSCPTR_PRV` register is used as a previous descriptor pointer overrun (PDPO) status bit. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error occurs when used for descriptor fetches. As a result, bit 1 and 0 of the `DMA_DSCPTR_PRV` register can be used for status. For more information, see the section on descriptor pointer capture in the DMA functional description.

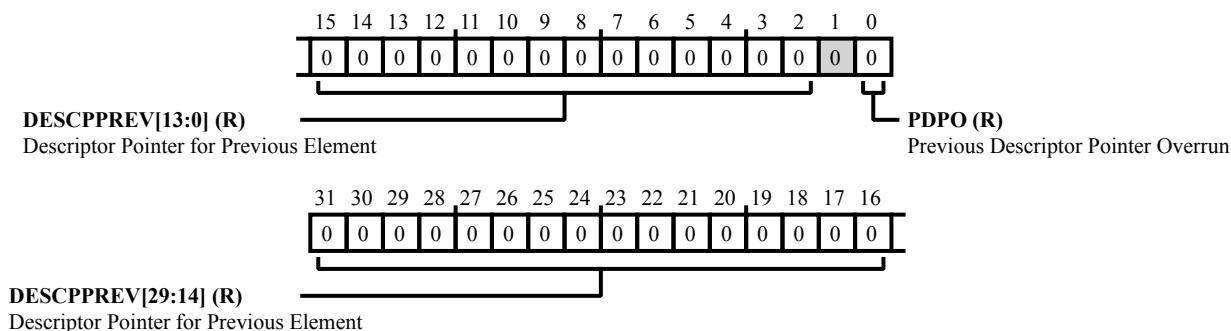


Figure 27-14: `DMA_DSCPTR_PRV` Register Diagram

Table 27-24: `DMA_DSCPTR_PRV` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/NW)	DESCPPREV	Descriptor Pointer for Previous Element. The <code>DMA_DSCPTR_PRV.DESCPPREV</code> bit field contains the initial descriptor pointer for the previous work unit.
0 (R/NW)	PDPO	Previous Descriptor Pointer Overrun. The <code>DMA_DSCPTR_PRV.PDPO</code> bit signifies an alignment error. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error would occur when used for descriptor fetches.
		0 No Error Occurred
		1 Error Occurred

Status Register

The `DMA_STAT` register indicates the status of DMA work units, the FIFO, errors, interrupts, and triggers.

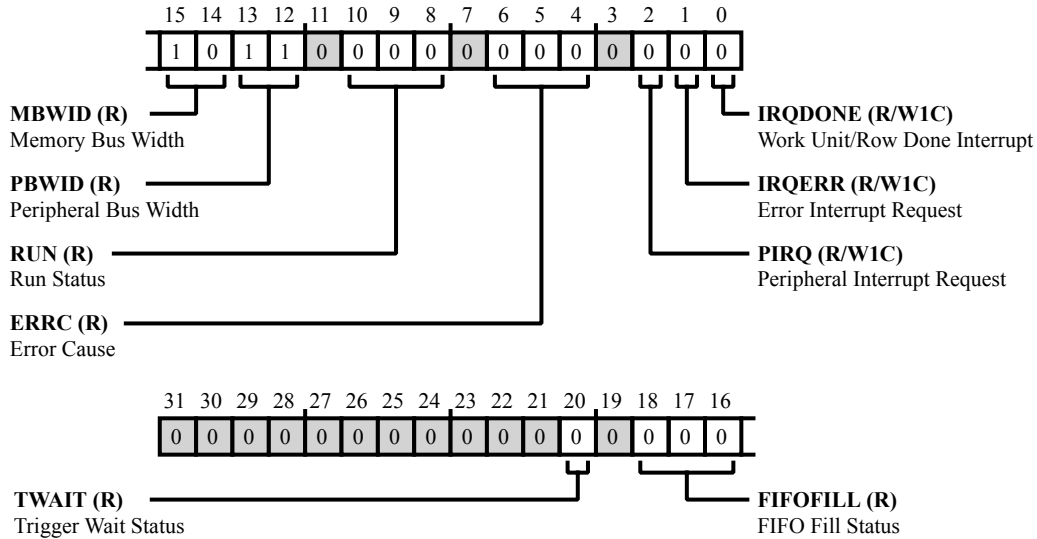


Figure 27-15: `DMA_STAT` Register Diagram

Table 27-25: `DMA_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	TWAIT	Trigger Wait Status. The <code>DMA_STAT.TWAIT</code> bit indicates whether the DMA has or has not received a trigger. This bit is set until it reaches the next wait state. At that point, the bit is cleared, the DMA stops processing that work unit, and the following work unit processes. The DMA does not distinguish between one or more triggers received.
		0 No Trigger Received
		1 Trigger Received
18:16 (R/NW)	FIFOFILL	FIFO Fill Status. The <code>DMA_STAT.FIFOFILL</code> bit field reports the quantity of data in the FIFO relative to available space.
		0 Empty
		1 Empty < FIFO = 1/4 Full
		2 1/4 Full < FIFO = 1/2 Full
		3 1/2 Full < FIFO = 3/4 Full
		4 3/4 Full < FIFO = Full
		5 Reserved

Table 27-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		6 Reserved
		7 Full
15:14 (R/NW)	MBWID	Memory Bus Width. The DMA_STAT.MBWID bit field indicates the width of the memory bus connected to this DMA.
		0 2 Bytes
		1 4 Bytes
		2 8 Bytes
		3 16 Bytes
13:12 (R/NW)	PBWID	Peripheral Bus Width. The DMA_STAT.PBWID bit field indicates the width of the peripheral bus connected to this DMA.
		0 1 Byte
		1 2 Bytes
		2 4 Bytes
		3 8 Bytes
10:8 (R/NW)	RUN	Run Status. The DMA_STAT.RUN bit field reports the DMA's current operational state. If the DMA is in idle or stop state, the DMA_CFG.EN bit is either 0 or 1. This bit field does not clear when a transition of the DMA_CFG.EN bit from 0 to 1 occurs. The DMA_STAT.RUN clears automatically when the DMA completes.
		0 Idle/Stop State
		1 Descriptor Fetch
		2 Data Transfer
		3 Waiting for Trigger
		4 Waiting for Write ACK/FIFO Drain to Peripheral
		5 Reserved
		6 Reserved
		7 Reserved
6:4 (R/NW)	ERRC	Error Cause. When an interrupt request error occurs (DMA_STAT.IRQERR), the DMA updates the DMA_STAT.ERRC bit field to identify the type of error. For more information, see the errors section of the DMA functional description.

Table 27-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		0 Configuration Error
		1 Illegal Write Occurred While Channel Running
		2 Address Alignment Error
		3 Memory Access or Fabric Error
		4 Reserved
		5 Trigger Overrun
		6 Bandwidth Monitor Error
		7 Reserved
2 (R/W1C)	PIRQ	<p>Peripheral Interrupt Request.</p> <p>The DMA_STAT.PIRQ bit indicates a peripheral interrupt request. Programs can use the DMA_STAT.PIRQ bit status to help determine which DMA asserted the interrupt request. This bit also helps to distinguish between an interrupt request caused by the state of the work unit and an interrupt request caused by the peripheral.</p>
		0 No Interrupt request
		1 Interrupt Request signaled by peripheral
1 (R/W1C)	IRQERR	<p>Error Interrupt Request.</p> <p>The DMA_STAT.IRQERR bit indicates that the DMA has detected a documented rule violation during DMA programming or operation. The DMA cannot, however, flag all possible programming or operation issues to indicate errors. Programmers can use the DMA_STAT.IRQERR bit to help determine which DMA issued the error interrupt request. The DMA_STAT.IRQERR does not clear a transition of the DMA_CFG.EN bit from 0 to 1. Clear the DMA_STAT.IRQERR bit with a write-1-to-clear operation prior to the DMA_CFG.EN transition for the fields to be reset.</p>
		0 No Error
		1 Error Occurred
0 (R/W1C)	IRQDONE	<p>Work Unit/Row Done Interrupt.</p> <p>The DMA_STAT.IRQDONE bit indicates that the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt request. Programs can use the DMA_STAT.IRQDONE status to help determine which DMA asserted the interrupt request. Programs can also use these bits to help distinguish between an interrupt request based on the state of the work unit and an interrupt request made by the peripheral. For more information, see the interrupts section of the DMA functional description.</p>
		0 Inactive
		1 Active

Inner Loop Count Start Value Register

For 2D DMA, the `DMA_XCNT` register contains the inner loop count. This value selects the number of `DMA_CFG.MSIZE` size data transfers that make up the length of a row. For 1D DMA, the `DMA_XCNT` register specifies the number of `DMA_CFG.MSIZE` size data transfers for the entire work unit. The `DMA_XCNT` register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

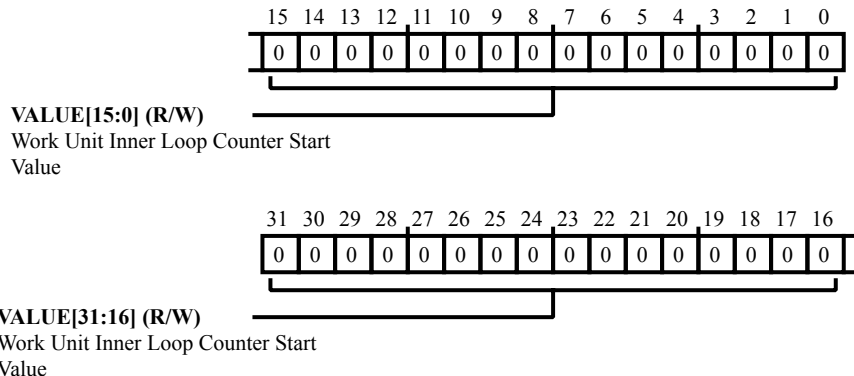


Figure 27-16: DMA_XCNT Register Diagram

Table 27-26: DMA_XCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Start Value. For 2D DMA, the <code>DMA_XCNT.VALUE</code> bit field contains the inner loop count. For 1D DMA, <code>DMA_XCNT.VALUE</code> specifies the number of <code>DMA_CFG.MSIZE</code> size data transfers for the entire work unit.

Current Count (1D) or Intra-row XCNT (2D) Register

For 1D DMA, the DMA loads the `DMA_XCNT_CUR` register from the `DMA_XCNT` register at the beginning of each work unit. For 2D DMA, the DMA loads the `DMA_XCNT_CUR` register from the `DMA_XCNT` register after the end of each row. The DMA decrements the value in `DMA_XCNT_CUR` register each time a `DMA_CFG.MSIZE` size data transfer occurs. When the count in `DMA_XCNT_CUR` register expires, the work unit is complete. In 2D DMA, the `DMA_XCNT_CUR` value is 0 only when the entire transfer is complete.

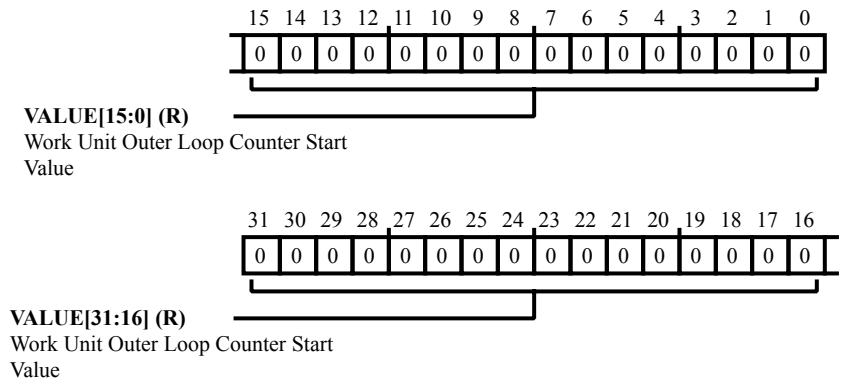


Figure 27-17: DMA_XCNT_CUR Register Diagram

Table 27-27: DMA_XCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Start Value. For 1D DMA, the <code>DMA_XCNT_CUR.VALUE</code> bit field holds the <code>DMA_XCNT</code> value at the beginning of each work unit. For 2D DMA, the <code>DMA_XCNT_CUR.VALUE</code> bit field holds the <code>DMA_XCNT</code> value after the end of each row.

Inner Loop Address Increment Register

The `DMA_XMOD` register contains a signed, two's-complement byte address increment. In 1D DMA, this increment is the stride that is applied after each `DMA_CFG.MSIZE` size data transfer. The `DMA_XMOD` register is read/write prior to enabling the channel, but is read-only after enabling the channel.

The `DMA_XMOD` register value is specified in bytes, regardless of the work unit size. In 2D DMA, this increment is applied after each `DMA_CFG.MSIZE` size data transfer in the inner loop, up to but not including the last `DMA_CFG.MSIZE` size data transfer in each inner loop. After the last `DMA_CFG.MSIZE` size data transfer in each inner loop, the `DMA_YMOD` register is applied instead, including the last `DMA_CFG.MSIZE` size data transfer of a work unit.

The `DMA_XMOD` field can be set to 0. In this case, DMA is performed repeatedly to or from the same address. This approach can be useful for transferring data between a data register and an external memory-mapped peripheral.

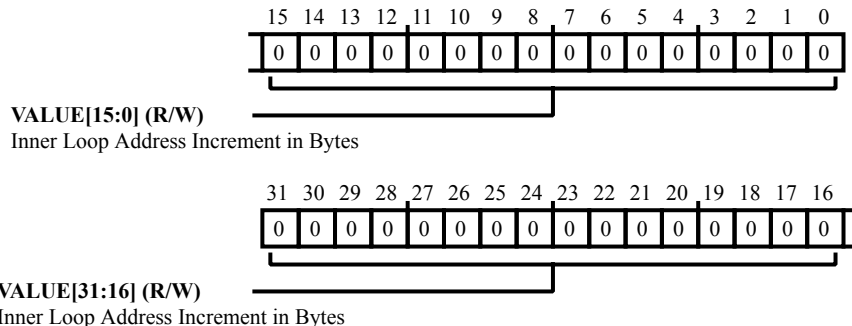


Figure 27-18: `DMA_XMOD` Register Diagram

Table 27-28: `DMA_XMOD` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Loop Address Increment in Bytes. The <code>DMA_XMOD.VALUE</code> bit field contains a signed, two's-complement byte address increment.

Outer Loop Count Start Value (2D only) Register

For 2D DMA, the `DMA_YCNT` register contains the outer loop count. This register is not used in 1D DMA mode. The `DMA_YCNT` register specifies the number of rows in the outer loop of a 2D DMA sequence. The `DMA_YCNT` register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

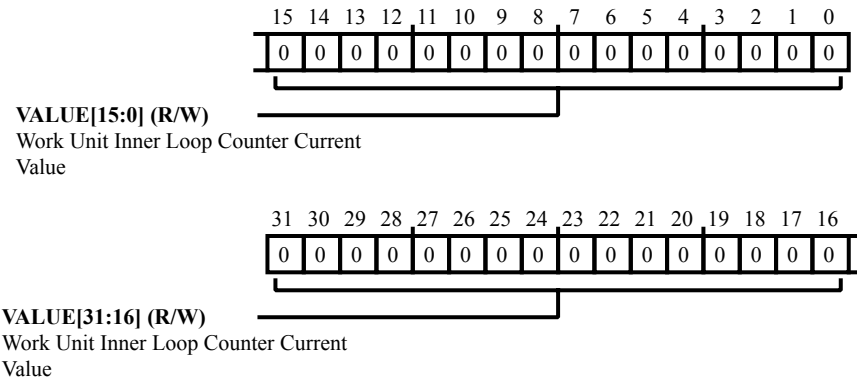


Figure 27-19: `DMA_YCNT` Register Diagram

Table 27-29: `DMA_YCNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Current Value. For 2D DMA, the <code>DMA_YCNT.VALUE</code> bit field contains the outer loop count.

Current Row Count (2D only) Register

For 2D DMA, the DMA loads the `DMA_YCNT_CUR` register from the `DMA_YCNT` register at the beginning of each 2D DMA session. The `DMA_YCNT_CUR` register is not used for 1D DMA. The DMA decrements the `DMA_YCNT_CUR` register each time the `DMA_XCNT_CUR` register expires during 2D DMA operation, signifying the completion of an entire row transfer.

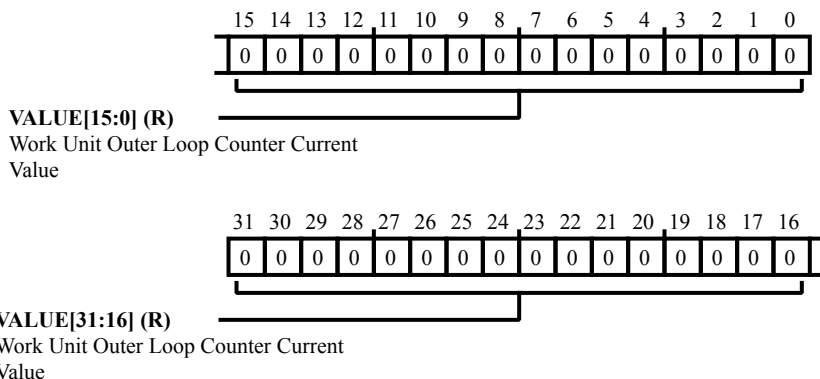


Figure 27-20: `DMA_YCNT_CUR` Register Diagram

Table 27-30: `DMA_YCNT_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Current Value. For 2D DMA, the <code>DMA_YCNT_CUR.VALUE</code> bit field holds the value from the <code>DMA_YCNT</code> register at the beginning of each 2D DMA session.

Outer Loop Address Increment (2D only) Register

The `DMA_YMOD` register contains a signed, two's-complement value. This byte address increment is applied after each decrement of the `DMA_YCNT_CUR` register. The value is the offset between the last word of one row and the first word of the next row. Note that `DMA_YMOD` is specified in bytes, regardless of the work unit size. The `DMA_YMOD` register is read/write prior to enabling the channel, but is read-only after enabling the channel.

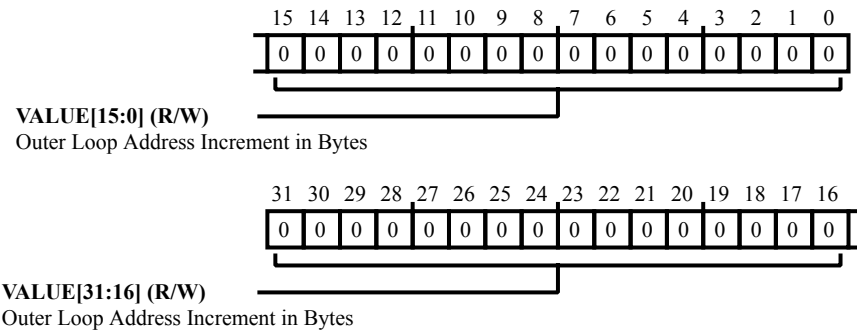


Figure 27-21: `DMA_YMOD` Register Diagram

Table 27-31: `DMA_YMOD` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Outer Loop Address Increment in Bytes. The <code>DMA_YMOD.VALUE</code> bit field contains a signed, two's-complement value.

28 Extended Memory DMA (EMDMA)

The Extended Memory DMA engine can be used in applications that copy data in a non-sequential manner. This includes delay lines, scatter and gather, and circular access types.

NOTE: Previous SHARC processors featured external port DMA. Current SHARC products use EMDMA that can access all memory locations (L1/L2/L3) for source and destination DMA operations.

EMDMA Features

EMDMA frees the processor core, allowing it to perform other operations while the data transfers between memories occurs as a background task.

EMDMA has the following features and capabilities.

- Standard mode DMA Transfer
- Chained mode with direction on-the-fly
- Tap list mode (scatter/gather)
- Delay line mode (write to read)
- All the DMA modes can operate in circular fashion
- In circular operation, some modes allow a write back of the index pointer for correct addressing of the next Transfer Control Block (TCB)

EMDMA Functional Description

The following sections provide information on the functional operations and operating modes of EMDMA.

Internal-to-internal DMA

This is accomplished by indexing all external parameter registers with internal addresses.

DMA bursting

DMA supports burst transfers only when the modifier is 1, for any other modifier values, single accesses are performed. The burst size is not user configurable and is chosen by the processor for optimal performance. The maximum burst size is 8 and the minimum is 1.

Transfer control blocks

The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously re-run the same DMA.

Chain pointer DMA

In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete.

ADSP-2156x EMDMA Register List

The EMDMA controllers support a variety of direct memory access operations which can access any system memory and transfer the entire block of data. A set of registers govern EMDMA operations. For more information on EMDMA functionality, see the EMDMA register descriptions.

Table 28-1: ADSP-2156x EMDMA Register List

Name	Description
EMDMA_BASE	External Base Address Register
EMDMA_BUFLLEN	Circular Buffer Length Register
EMDMA_CHNPTR	Chain Pointer Register
EMDMA_CNT0	Internal Count Register
EMDMA_CNT1	External Count Register
EMDMA_CTL	External Memory DMA Control Register
EMDMA_INDX0	Internal Index Register
EMDMA_INDX1	External Index Register
EMDMA_MOD0	Internal Modifier Register
EMDMA_MOD1	External Modifier Register
EMDMA_TCNT	Delay Line Tap Count Register
EMDMA_TPTR	Tap List Pointer Register

ADSP-2156x EMDMA Interrupt List

Table 28-2: ADSP-2156x EMDMA Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
130	EMDMA0_DONE	EMDMA0 Transfer Done	Edge	
131	EMDMA1_DONE	EMDMA1 Transfer Done	Edge	

ADSP-2156x EMDMA Trigger List

Table 28-3: ADSP-2156x EMDMA Trigger List Masters

Trigger ID	Name	Description	Sensitivity
57	EMDMA0_DONE	EMDMA0 EMDMA0 DMA Done	Edge
58	EMDMA1_DONE	EMDMA1 EMDMA1 DMA Done	Edge

Table 28-4: ADSP-2156x EMDMA Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

DMA Addressing

Besides the traditional internal-to-external addressing type, the DMA module also supports internal-to-internal transfers. This is accomplished by indexing all external parameter registers with internal addresses. The DMA controller recognizes the transfer by addresses and not by an additional control bit setting.

All the DMA addresses given by EMDMA are word-aligned byte addresses. The programming for the index registers is provided in the *Register Descriptions* section.

DMA Burst Transfers

DMA supports burst transfers only when the modifier is 1. For any other modifier values, single accesses are performed.

The burst size is not user-configurable and is chosen by the processor for optimal performance. The maximum burst size is 8 and the minimum is 1.

The EMDMA uses appropriate burst transfer sizes for optimal throughput. For example, if the word count is 15, then 5 bursts are performed with burst sizes of 8+4+1+1+1 transfers.

Transfer Control Block (TCB) Memory Storage

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the processor's internal memory. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Assignment

The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously rerun the same DMA. The EMDMA reads each word of the TCB and loads it into the corresponding register. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

The address field of the chain pointer registers is only 30 bits wide. If a program writes a symbolic address to bit 30 of the chain pointer there may be a conflict with the `EMDMA_CHNPTR.PCI` bit. Programs should clear the upper bits of the address then AND the `EMDMA_CHNPTR.PCI` bit separately, if needed.

Starting Chain Loading

A DMA sequence is defined as the sum of the DMA transfer from when the parameter registers initialize to when the count register decrements to zero. The EMDMA module has a chaining enable bit (`EMDMA_CTL.CHEN`).

To start the chain, write the internal (channel 0) index address of the first TCB to the chain pointer register (`EMDMA_CHNPTR`). When chaining is enabled, DMA transfers are initiated by writing a memory address to the chain pointer register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

During TCB chain loading, the EMDMA loads the DMA channel parameter registers with values retrieved from system memory.

The address in the chain pointer register points to the highest address of the TCB. This contains the internal (channel 0) index parameter. This means that if a program declares an array to hold the TCB, the chain pointer register should point to the last location of the array and not to the first TCB location.

Buffered Chain Loading Register

The chain pointer register (`EMDMA_CHNPTR`) is buffered. Before the chain loading starts, the buffer is copied into the chain pointer register and is decremented after each register is loaded.

The chain pointer register can be loaded with a new address at any time during the DMA sequence (`EMDMA_CTL.CHEN=1`). This allows a DMA channel to have chaining status deactivated (chain pointer register = 0x0) until some event occurs that loads the chain pointer register with a non-zero value. Writing all zeros to the address field of the chain pointer register also deactivates chaining for the next TCB.

TCB Storage

The EMDMA supports several types of DMA, resulting in different lengths of TCBs. The TCB size varies from six locations (chained DMA) to 13 locations (delay line DMA). The *EMDMA TCBs* table shows the required TCBs for chained DMA.

In the following tables, TCB refers to the start address of the TCB array.

Table 28-5: EMDMA TCBs for Standard DMA

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_MOD1
TCB + 0x2	EMDMA_INDX1
TCB + 0x3	EMDMA_CNT0
TCB + 0x4	EMDMA_MOD0
TCB + 0x5	EMDMA_INDX0

The order the descriptors are fetched with circular buffering enabled is shown in the *EMDMA TCBs for Standard Circular DMA* table.

Table 28-6: EMDMA TCBs for Standard Circular DMA

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_BUFLFN
TCB + 0x2	EMDMA_BASE
TCB + 0x3	EMDMA_MOD1
TCB + 0x4	EMDMA_INDX1
TCB + 0x5	EMDMA_CNT0
TCB + 0x6	EMDMA_MOD0
TCB + 0x7	EMDMA_INDX0

For delay line DMA, TCB loading is split into two sequences to improve overall priority. The first TCB loads the write parameters (EMDMA_INDX0 – EMDMA_BUFLFN) and the second loads the read parameters (EMDMA_INDX0 – EMDMA_CHNPTR). This two stage loading is transparent to the application. The order the descriptors are fetched for delay line DMA, as shown in the *EMDMA TCBs for Delay Line DMA* table.

Table 28-7: EMDMA TCBs for Delay Line DMA

Address	Register
Delay Line Read	
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_TPTR
TCB + 0x2	EMDMA_TCNT
TCB + 0x3	EMDMA_MOD1
TCB + 0x4	EMDMA_CNT0

Table 28-7: EMDMA TCBs for Delay Line DMA (Continued)

Address	Register
TCB + 0x5	EMDMA_INDX0
Delay Line Write	
TCB + 0x6	EMDMA_BUFLLEN
TCB + 0x7	EMDMA_BASE
TCB + 0x8	EMDMA_MOD1
TCB + 0x9	EMDMA_INDX1
TCB + 0xA	EMDMA_CNT0
TCB + 0xB	EMDMA_MOD0
TCB + 0xC	EMDMA_INDX0

The order the descriptors are fetched for scatter/gather DMA with circular buffering disabled is shown in the *EMDMA TCBs for Scatter/Gather DMA* table.

Table 28-8: EMDMA TCBs for Scatter/Gather DMA

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_TPTR
TCB + 0x2	EMDMA_TCNT
TCB + 0x3	EMDMA_MOD1
TCB + 0x4	EMDMA_INDX1
TCB + 0x5	EMDMA_CNT0
TCB + 0x6	EMDMA_MOD0
TCB + 0x7	EMDMA_INDX0

The order the descriptors are fetched for scatter/gather DMA with circular buffering enabled is shown in the *EMDMA TCBs for Circular Scatter/Gather DMA* table.

Table 28-9: EMDMA TCBs for Circular Scatter/Gather DMA

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_BUFLLEN
TCB + 0x2	EMDMA_BASE
TCB + 0x3	EMDMA_TPTR
TCB + 0x4	EMDMA_TCNT

Table 28-9: EMDMA TCBs for Circular Scatter/Gather DMA (Continued)

Address	Register
TCB + 0x5	EMDMA_MOD1
TCB + 0x6	EMDMA_INDX1
TCB + 0x7	EMDMA_CNT0
TCB + 0x8	EMDMA_MOD0
TCB + 0x9	EMDMA_INDX0

EMDMA Operating Modes

This section and the *EMDMA_CTL Register Bit to Operating Modes* table show the different DMA modes which can be used. The complete register bit descriptions are in the *External Memory DMA Control Registers (EMDMA_CTL)*.

Table 28-10: EMDMA_CTL Register Bit to Operating Modes

Bit (Name)	Standard	Chained	Scatter/Gather	Delay Line
Control Bits				
0 (EN)			Valid	
1 (TRAN)		Valid		N/A
2 (CHEN)			Valid	
3 (DLEN)		N/A		Valid
4 (CBEN)			Valid	
5 (DFLSH)			Valid	
6			N/A	
7 (WRBEN)	N/A	Valid	(=0)	(=1)
8 (OFCEN)		Valid		N/A
9 (TLEN)		N/A	Valid	N/A
11–10			N/A	
12 (INTDONE0)			Valid	
15–13			N/A	
Status Bits				
17–16 (DFS)			Valid	
19–18			N/A	
20 (DMAS0)			Valid	
21 (CHS)	N/A	Valid	N/A	Valid

Table 28-10: EMDMA_CTL Register Bit to Operating Modes (Continued)

Bit (Name)	Standard	Chained	Scatter/Gather	Delay Line
22 (TLS)	N/A		Valid	N/A
23 (WBS)	N/A			Valid
24 (DMAS1)	Valid			
25 (DIRS)	Valid			
31–26	N/A			

NOTE: Reading additional bit-field information from N/A (Not applicable) bits does not generate a meaningful result.

A program sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers.

NOTE: The `EMDMA_CNT1` parameter register (read only) is a copy of the `EMDMA_CNT0` register. If `EMDMA_CNT0` is written, the `EMDMA_CNT1` register is updated automatically.

Standard DMA

A standard DMA (once it is configured) transfers data from location A to location B. An interrupt can be used to indicate the end of the transfer. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit (`EMDMA_CTL.EN`), write new parameters to the index, modify, and count registers (parameter registers), then set the DMA enable bit to re-enable DMA.

This DMA type resembles the traditional DMA type to initialize the different internal and external parameters (channel0 and channel1) (index, modify and count) registers and configuration of the DMA control registers.

Circular Buffered DMA

Circular buffered DMA resembles the traditional core DAG circular buffered mode by using registers for circular buffering. In this mode, the DMA needs two additional registers (base and length) to support reads and writes to a circular buffer. The *Circular Buffering Write DMA* and *Circular Buffering Read DMA* figures illustrate circular buffered DMA, in contrast with the *Standard Write DMA* figure.

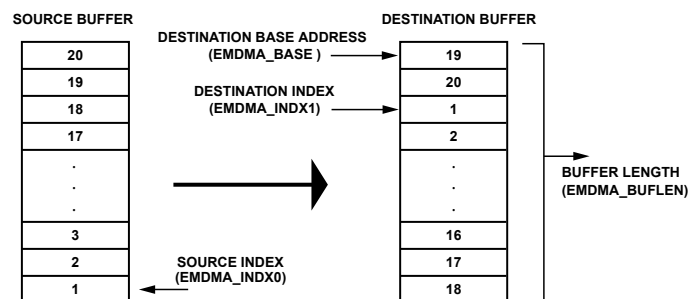


Figure 28-1: Circular Buffering Write DMA

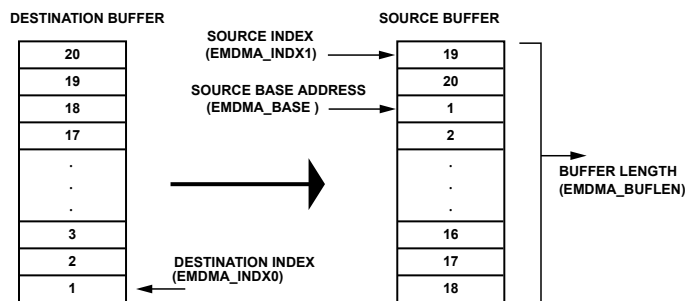


Figure 28-2: Circular Buffering Read DMA

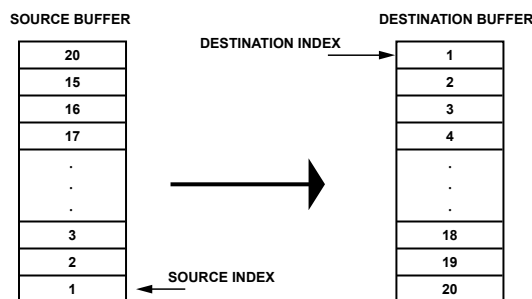


Figure 28-3: Standard Write DMA

NOTE: Circular buffering is available for all operating modes (standard, chained, tap list and delay line DMA).

Chained DMA Mode

Chained DMA sequences are a set of multiple DMA operations, each auto-initializing the next in line. It is used to support automated access by a linked list (repetitive reads and writes to a defined location defined by the individual TCBs). To start a new DMA sequence after the current one is finished, the EMDMA automatically loads new index, modify, and count values pointed to by that channel's `EMDMA_CHNPTR` register. Using chaining, programs can set up consecutive DMA operations and each operation can have different attributes.

DMA data transfers can be set up as continuous or periodic. With chained DMA, the attributes of a specific DMA are stored in internal memory and are referred to as a Transfer Control Block or TCB. Extended Memory DMA loads these attributes in chains for execution. This allows for multiple chains that are finite or infinite.

NOTE: When chaining is enabled, polling should not be used to determine DMA status only because the DMA appears inactive if it is sampled while the next TCB is loading. In such cases where chaining is enabled, along with the polling of DMA status bit, polling of the chaining status bit should also occur so that the correct status of the DMA is known. For example, the `EMDMA_CTL.CHS` bit should be polled as well as the `EMDMA_CTL.DMAS0` and `EMDMA_CTL.DMAS1` bits when EMDMA is configured in DMA chaining mode.

Data Direction On-the-Fly

A change of external memory data direction for each individual TCB in a chain sequence is allowed.

The `EMDMA_CHNPTR.CPDR` bit changes the data flow direction. If the `EMDMA_CHNPTR.CPDR = 0`, writes through channel 0 are performed; if `EMDMA_CHNPTR.CPDR = 1`, channel 0 reads are performed. This works similarly to the `EMDMA_CHNPTR.PCI` bit. The `EMDMA_CTL.OFCEN` and `EMDMA_CTL.CHEN` bits must be set (=1) to enable this functionality.

NOTE: If chaining is enabled with the `EMDMA_CTL.OFCEN = 1`, then the `EMDMA_CTL.TRAN` bit has no effect, and the data direction is determined by the `EMDMA_CHNPTR.CPDR` bit.

Write Back Circular Index Pointer

Operating the DMA in circular mode requires some special considerations. The index pointer of start address within the buffer may wrap around for the case when $IC \times IM > EL$ or it does not finish if $IC \times IM < EL$ where:

IC is the value of the `EMDMA_INDX0` register

IM is the value of the `EMDMA_MOD0` register

EL is the value of the `EMDMA_BUFLLEN` register

In both cases, the TCB start address is no longer valid.

Setting the `EMDMA_CTL.WRBEN` bit writes (at the end of current TCB block) the current index address + 1 into the TCB memory which is the start address for the next TCB. This bit is only selectable for chained DMA mode. For tap list and delay line modes, this bit is hardwired to 0 or 1.

Scatter/Gather DMA

The purpose of scatter/gather DMA is the transfer of data from/to non-contiguous memory blocks.

The scatter/gather DMA type is a fixed block size scatter/gather DMA that relies on tap list entries in system memory to calculate the (Channel 1 DMA Address) to scatter/gather the DMA. If the DMA direction is Channel 1 write (`EMDMA_CTL.TRAN = 1`) then it is a scatter DMA. If `EMDMA_CTL.TRAN = 0` then it is a gather DMA. This mode also supports chained and circular buffer chained DMAs.

See the *Scatter DMA (Writes)*, *Gather DMA (Reads)*, *Circular Buffering Scatter DMA (Writes)*, and *Circular Gather DMA (Reads)* figures.

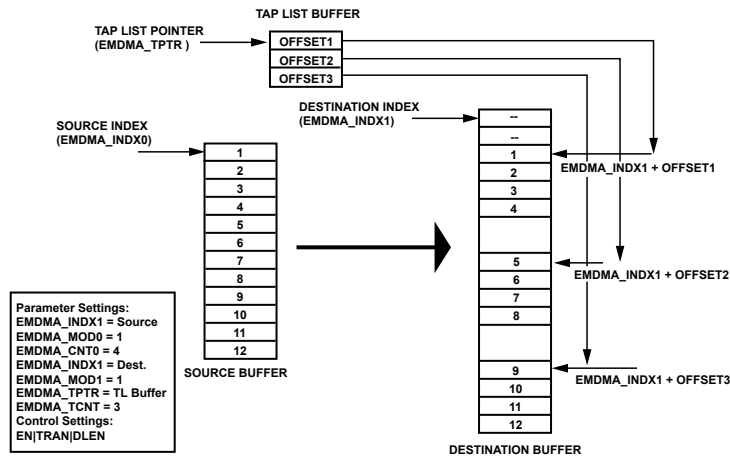


Figure 28-4: Scatter DMA (Writes)

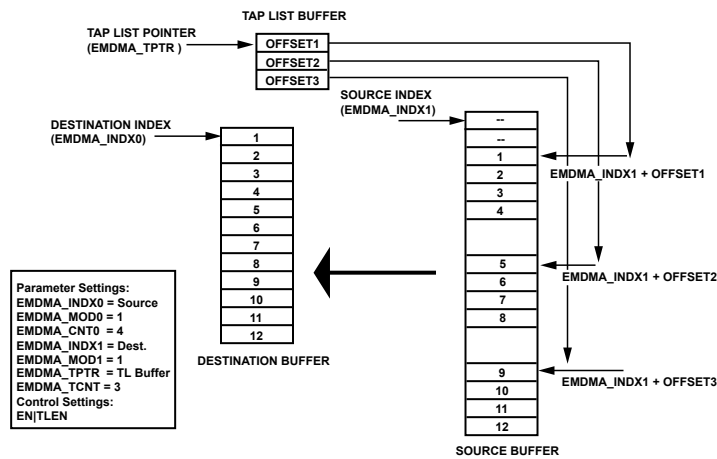


Figure 28-5: Gather DMA (Reads)

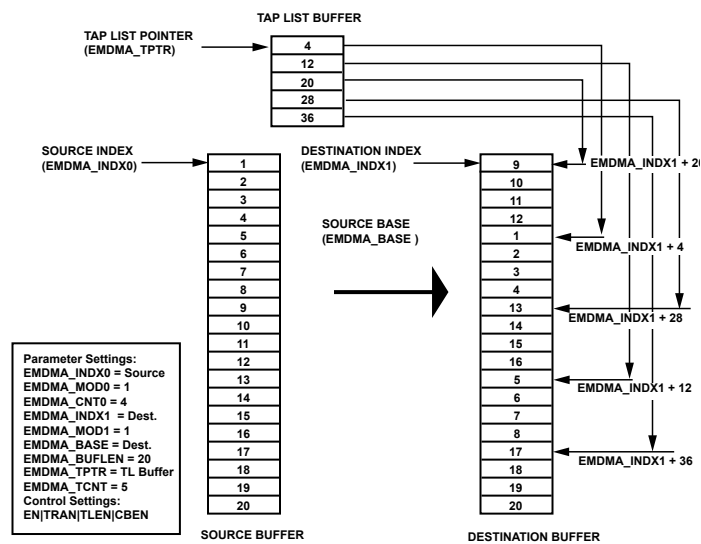


Figure 28-6: Circular Buffering Scatter DMA (Writes)

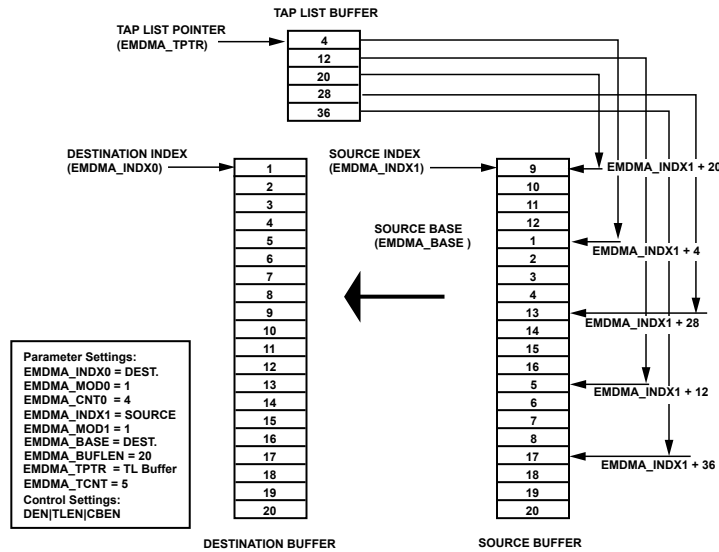


Figure 28-7: Circular Gather DMA (Reads)

For each 32-bit tap read, the Channel 1 read index is shown in the *Read/Write Index Pre-Modify (Scatter/Gather DMA)* table. Note that one tap list entry starts multiple reads.

Table 28-11: Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation	Result	
	Block Size	Tap
$EMDMA_INDX1 + EDMDA_TPTR[EMDMA_TCNT] + (EMDMA_MOD1 \times EDMDA_CNT0)$	N	0
$EMDMA_INDX1 + EDMDA_TPTR[0] + EDMDA_MOD1 \times 1$		
$EMDMA_INDX1 + EDMDA_TPTR[0] + EDMDA_MOD1 \times 2$		
$EMDMA_INDX1 + EDMDA_TPTR[0] + EDMDA_MOD1 \times 3$		
$EMDMA_INDX1 + EDMDA_TPTR[0] + EDMDA_MOD1 \times N$	N	1
$EMDMA_INDX1 + EDMDA_TPTR[1] + EDMDA_MOD1 \times 1$		
$EMDMA_INDX1 + EDMDA_TPTR[1] + EDMDA_MOD1 \times 2$		
$EMDMA_INDX1 + EDMDA_TPTR[1] + EDMDA_MOD1 \times 3$		
$EMDMA_INDX1 + EDMDA_TPTR[1] + EDMDA_MOD1 \times N$	N	M
$EMDMA_INDX1 + EDMDA_TPTR[M] + EDMDA_MOD1 \times 1$		
$EMDMA_INDX1 + EDMDA_TPTR[M] + EDMDA_MOD1 \times 2$		
$EMDMA_INDX1 + EDMDA_TPTR[M] + EDMDA_MOD1 \times 3$		
$EMDMA_INDX1 + EDMDA_TPTR[M] + EDMDA_MOD1 \times N$		

Pre Modified Read/Write Index

For scatter/gather DMA, the tap list modifiers are employed and the number of taps is determined by the tap list count register ($EMDMA_TCNT$). The number of sequential reads (block size) from every tap is determined by the

internal count register (`EMDMA_CNT0`), and is the same for every tap. The read/write pointer in external index register (`EMDMA_INDX1`) serves as the index address for these read/writes.

`TL[N]` is the first tap list entry in the memory as pointed to by the tap list pointer register (`EMDMA_TPTR`). The tap list entries are 27-bit signed integers. For each read/write block, the DMA state machine fetches the offset from the tap list. The offset is added to the `EMDMA_INDX1` register value to get the start address of the next block. The Channel 1 addresses are circular buffered if circular buffering is enabled (see the *Circular Buffering Scatter DMA (Writes)* and *Circular Buffering Gather DMA (Reads)* figures in *Scatter/Gather DMA*).

Once the `EMDMA_CNT0` register for the final tap decrements to zero (both `EMDMA_TCNT` and `EMDMA_CNT0` are zero), then the tap list DMA access is complete and the DMA completion interrupt is generated (if chaining is enabled the interrupt depends on the `EMDMA_CHNPTR.PCI` bit setting).

The write back mode (`EMDMA_CTL.WRBEN` bit) is hardwired to zero for tap list based DMA (as the addressing is pre-modify, and therefore the `EMDMA_INDX1` value coincides with the TCB value even at the end of the DMA).

Delay Line DMA

Delay line DMA is used to support reads and writes to delay line buffers with limited core interaction. In this sense, delay line DMA is a quantity of integrated writes followed by reads from channel 1 (a delay line DMA access). Delay line DMA is described in the following sections.

NOTE: Delay line DMA can only operate by using chained DMA mode (`EMDMA_CTL.CHEN` bit set). In order to use delay line DMA for a single DMA sequence, initialize the `EMDMA_CHNPTR` register to zero in the TCB.

NOTE: Delay line DMA can be used in any system memory.

The delay line DMA operation follows these steps:

1. Load the first half of TCB for writing (seven parameters).
2. DMA writes to the delay line buffer until `IC = 0`.
3. Update the EI index pointer if circular mode is enabled.
4. Load the second half of TCB for reading (six parameters).
5. DMA tap based reads from the delay line buffer until `RC = 0`.

Jump to step 1.

Writes to delay line. The number of writes is determined by the `EMDMA_CNT0` register. The data is fetched from the `EMDMA_INDX0` register and the `EMDMA_MOD0` register is used as the internal modifier. The `EMDMA_INDX1` register serves as the external index and is incremented by the `EMDMA_MOD1` register after each write. These writes are circular buffered if circular buffering is enabled. See the *Write to Delay Line Buffer* figure.

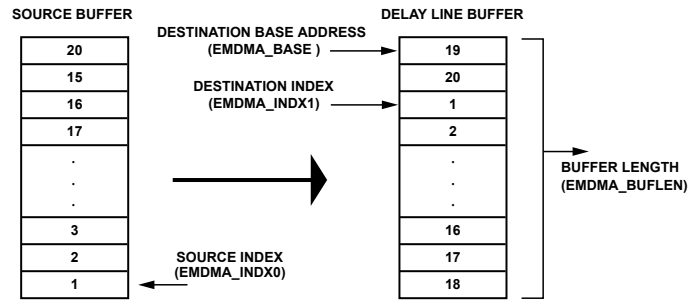


Figure 28-8: Write to Delay Line Buffer

When the writes are complete, ($EMDMA_CNT0 = 0$) the $EMDMA_INDX1$ register, which serves as the write pointer of the delay line, is written back ($EMDMA_CTL.WRBEN$ is hardwired to 1) to the TCB location from where it was fetched.

Reads from the delay line. For reads, the tap list (TL) modifiers are used and the number of reads is determined by the $EMDMA_CNT0$ register. The write pointer in the $EMDMA_INDX0$ register serves as the index address for these reads (reads start from where writes end). The $EMDMA_INDX0$ register, along with tap list modifiers, are used in a pre-modify addressing mode to create the external address for the reads. Therefore, for each read, the DMA controller fetches the external modifier ($EMDMA_TCNT$ register) from the tap list and the reads are circular buffered (if enabled). See the *Read From Delay Line Buffer* figure.

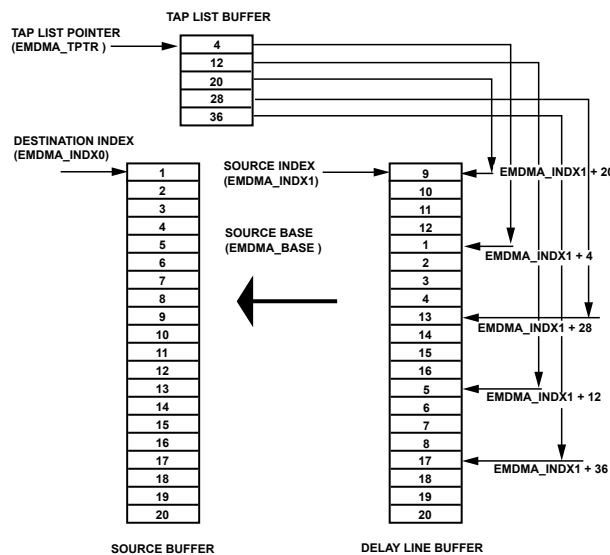


Figure 28-9: Read From Delay Line Buffer

Pre-Modified Read Index

Note that $TL[N]$ is the first tap list entry in memory pointed to by the tap list pointer register ($EMDMA_TPTR$). Tap list entries are 27-bit signed integers. Therefore, for each read-block, the DMA state machine fetches the offset external modifier from the tap list. The reads are circular buffered if circular buffering is enabled.

NOTE: The channel 1 DMA address generation follows pre-modify addressing for reads in delay line DMA and therefore the $EMDMA_INDX1$ register values are not updated. Also the $EMDMA_MOD1$ register does not

have any effect during these delay line reads. Once the read count completes, the `EMDMA_CNT0` register decrements to zero (both `EMDMA_CNT0` and `EMDMA_TCNT` are zero) for the final tap. Finally, the delay line DMA access completes and the DMA completion interrupt is generated. If chaining is enabled, the interrupt is dependent on the `EMDMA_CHNPTR.PCI` bit setting. The delay line DMA can only be initialized using the TCB. In order to use the delay line DMA for a single DMA sequence, initialize the `EMDMA_CHNPTR` register to zero in the TCB.

For each 32-bit tap read, the channel 1 read index is shown in the *Read/Write Index Pre-Modify (Scatter/Gather DMA)* table. Note that one tap list entry starts multiple reads.

Table 28-12: Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation	Result	
	Block Size	Tap
$EMDMA_INDX0 + EMDMA_TPTR[EMDMA_TCNT] + (EMDMA_MOD1 \times EMDMA_CNT0)$		
$EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 \times 1$	N	0
$EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 \times 2$		
$EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 \times 3$		
$EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 \times N$		
$EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 \times 1$	N	1
$EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 \times 2$		
$EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 \times 3$		
$EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 \times N$		
$EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 \times 1$	N	M
$EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 \times 2$		
$EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 \times 3$		
$EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 \times N$		

ADSP-2156x EMDMA Register Descriptions

Extended Memory DMA (EMDMA) contains the following registers.

Table 28-13: ADSP-2156x EMDMA Register List

Name	Description
<code>EMDMA_BASE</code>	External Base Address Register
<code>EMDMA_BUFLen</code>	Circular Buffer Length Register
<code>EMDMA_CHNPTR</code>	Chain Pointer Register
<code>EMDMA_CNT0</code>	Internal Count Register
<code>EMDMA_CNT1</code>	External Count Register

Table 28-13: ADSP-2156x EMDMA Register List (Continued)

Name	Description
EMDMA_CTL	External Memory DMA Control Register
EMDMA_INDX0	Internal Index Register
EMDMA_INDX1	External Index Register
EMDMA_MOD0	Internal Modifier Register
EMDMA_MOD1	External Modifier Register
EMDMA_TCNT	Delay Line Tap Count Register
EMDMA_TPTR	Tap List Pointer Register

External Base Address Register

The `EMDMA_BASE` register contains the external base address of the Delay Line buffer. This is used for maintaining circular buffered read/writes to the delay line.

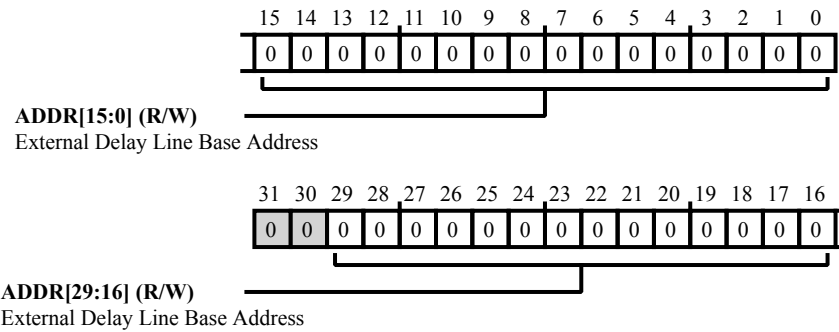


Figure 28-10: EMDMA_BASE Register Diagram

Table 28-14: EMDMA_BASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	ADDR	External Delay Line Base Address. The <code>EMDMA_BASE.ADDR</code> bit field contains the external base address of the Delay Line buffer.

Circular Buffer Length Register

The `EMDMA_BUFLEN` register holds the circular buffer length for the Delay-line DMA.

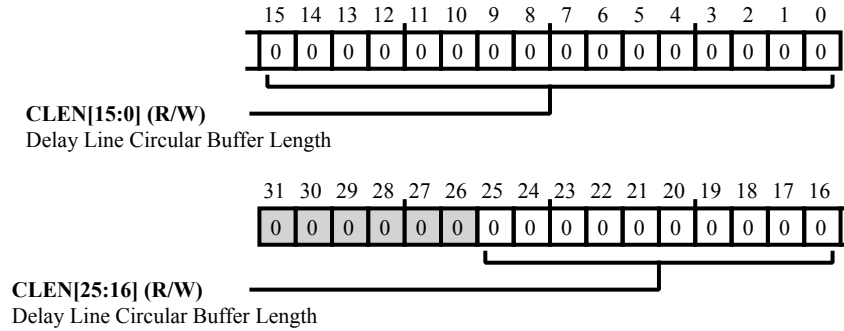


Figure 28-11: EMDMA_BUFLEN Register Diagram

Table 28-15: EMDMA_BUFLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:0 (R/W)	CLEN	Delay Line Circular Buffer Length. The <code>EMDMA_BUFLEN.CLEN</code> bit field holds the circular buffer length for the Delay-line DMA.

Chain Pointer Register

The `EMDMA_CHNPTR` register contains the address of the next descriptor in memory when the `EMDMA_CTL.CHEN` bit =1. This register also has bits to change DMA directions for each descriptor and the PCI bit. Note that the lower 30-bits of this register are to be written with 30 MSB's of the Word Aligned Byte addresses of the corresponding next descriptor address.

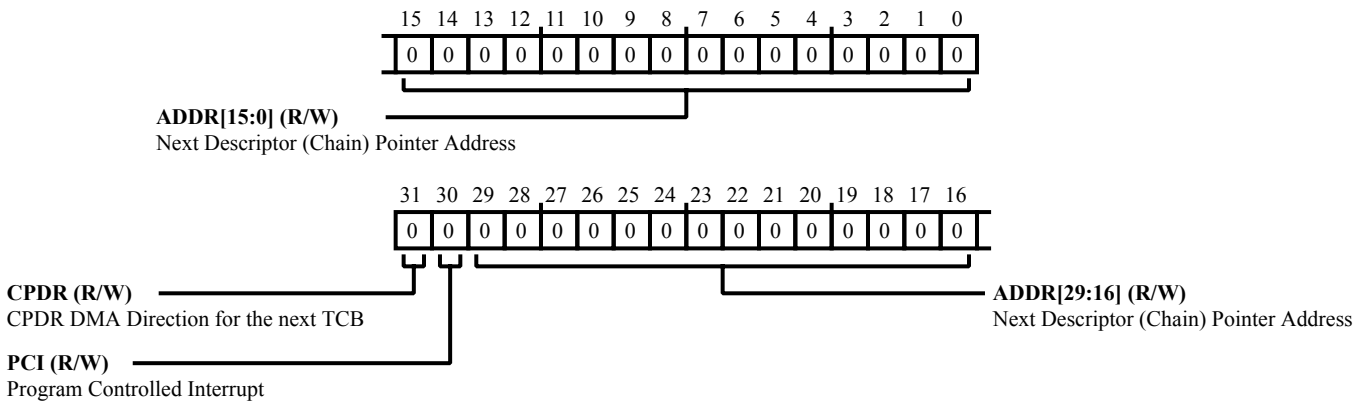


Figure 28-12: EMDMA_CHNPTR Register Diagram

Table 28-16: EMDMA_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CPDR	CPDR DMA Direction for the next TCB. The <code>EMDMA_CHNPTR.CPDR</code> bit configures whether the DMA is a write to internal memory or a read from internal memory. Note that this setting is applicable only if <code>EMDMA_CTL.OFCEN</code> =1 and is not applicable for Delay Line DMA.
		0 Write to Channel 0 (Channel 1 reads)
		1 Read from Channel 0 (Channel 1 Writes)
30 (R/W)	PCI	Program Controlled Interrupt. The <code>EMDMA_CHNPTR.PCI</code> bit PCI sets whether an interrupt is generated after the current TCB or if no interrupt is generated. (Only affects DMA if chaining is enabled)
		0 Enable DMA Channel interrupt to occur at the completion of the entire DMA chained transfer
		1 Enable DMA Channel interrupt to occur at the completion of current DMA sequence
29:0 (R/W)	ADDR	Next Descriptor (Chain) Pointer Address. The <code>EMDMA_CHNPTR.ADDR</code> bit field provides the address of the next descriptor in memory.

Internal Count Register

The `EMDMA_CNT0` register contains the number of words to be transferred for channel 0 DMA. Note: If Delay Line DMA is enabled then the `EMDMA_CNT0` register serves as the count register for the Delay Line writes.

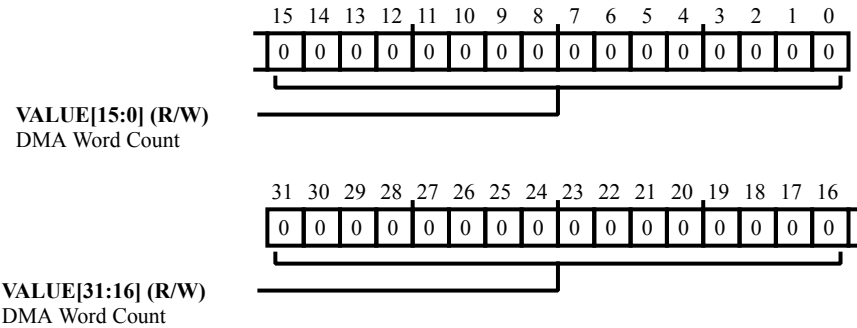


Figure 28-13: EMDMA_CNT0 Register Diagram

Table 28-17: EMDMA_CNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Word Count. The <code>EMDMA_CNT0.VALUE</code> bit field is the DMA word count.

External Count Register

The `EMDMA_CNT1` register contains the number of words to be transferred for channel 1 DMA. Note: If Delay Line DMA is enabled then the `EMDMA_CNT1` register serves as the count register for the Delay Line writes.

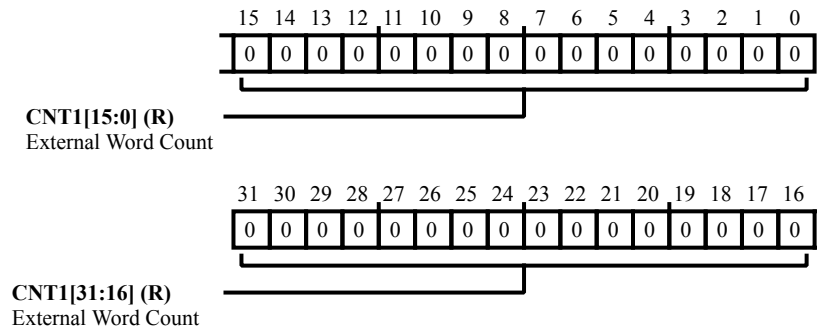


Figure 28-14: EMDMA_CNT1 Register Diagram

Table 28-18: EMDMA_CNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT1	External Word Count. The <code>EMDMA_CNT1 . CNT1</code> bit field is the external word count.

External Memory DMA Control Register

The `EMDMA_CTL` register contains bits that enable and configure EMDMA and indicate DMA transfer status.

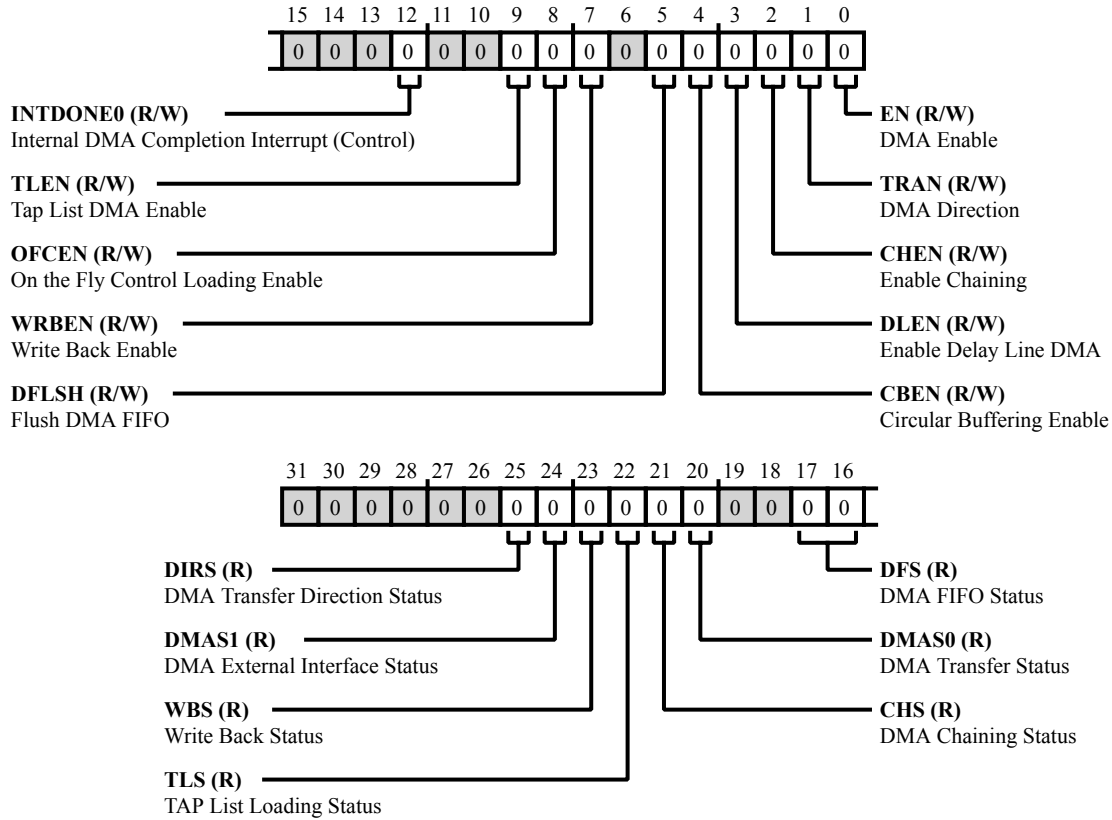


Figure 28-15: EMDMA_CTL Register Diagram

Table 28-19: EMDMA_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	DIRS	DMA Transfer Direction Status. The <code>EMDMA_CTL.DIRS</code> bit provides the DMA transfer status direction. This is useful for delay line DMA where the transfer direction changes with the state of the DMA state machine. For standard DMA, the <code>EMDMA_CTL.DIRS</code> bit reflects the state of the <code>EMDMA_CTL.TRAN</code> bit.
		0 DMA direction is Channel 1 Reads
		1 DMA direction is Channel 1 Writes

Table 28-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/NW)	DMAS1	DMA External Interface Status. The EMDMA_CTL.DMAS1 bit provides the DMA channel 1 transfer status.
		0 Channel 1 DMA does not have any access pending
		1 Channel 1 DMA access are pending
23 (R/NW)	WBS	Write Back Status. The EMDMA_CTL.WBS bit provides the delay line write pointer write back status.
		0 Write pointer write back is not active
		1 Write pointer write back is active
22 (R/NW)	TLS	TAP List Loading Status. The EMDMA_CTL.TLS bit provides the DMA tap list loading status.
		0 TAP list loading is not active
		1 TAP list loading is active
21 (R/NW)	CHS	DMA Chaining Status. The EMDMA_CTL.CHS bit provides the DMA chaining status.
		0 DMA chain loading is not active
		1 DMA chain loading is active
20 (R/NW)	DMAS0	DMA Transfer Status. The EMDMA_CTL.DMAS0 bit provides the DMA channel 0 transfer status.
		0 DMA idle
		1 DMA in progress
17:16 (R/NW)	DFS	DMA FIFO Status. The EMDMA_CTL.DFS bit field provides the DMA FIFO status.
		0 FIFO EMPTY
		1 FIFO Partially Full
		2 Reserved
		3 FIFO Full

Table 28-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	INTDONE0	Internal DMA Completion Interrupt (Control). The EMDMA_CTL.INTDONE0 bit configures when the DMA complete interrupt is generated. The EMDMA_CTL.INTDONE0 =1 setting is provided for backward compatibility with older SHARC processors.
		0 Interrupt on access completion (Channel 0 or Channel 1 DMA completion)
		1 Interrupt on Channel 0 DMA completion.
9 (R/W)	TLEN	Tap List DMA Enable. The EMDMA_CTL.TLEN bit enables scatter/gather tap list DMA.
		0 Disables the tap list based scatter/gather DMA
		1 Enables the tap list based scatter/gather DMA
8 (R/W)	OFCEN	On the Fly Control Loading Enable. The control bits in EMDMA_CHNPTR register are used to describe the next TCB behavior if the EMDMA_CTL.OFCEN bit is set and therefore the DMA controls can be changed from TCB to TCB. 0 = disables the control bits in the EMDMA_CHNPTR register 1 = Enables the control bits in the EMDMA_CHNPTR register. If chaining is enabled with EMDMA_CTL.OFCEN bit set then the EMDMA_CTL.TRAN bit has no effect, and direction is determined by EMDMA_CHNPTR.CPDR bit.
7 (R/W)	WRBEN	Write Back Enable. The EMDMA_CTL.WRBEN bit enables write back of the EIEP register after reads and or writes. Write back is automatically enabled for delay line DMA. WRBEN is applicable only if chaining is enabled (EMDMA_CTL.CHEN =1).
5 (R/W)	DFLSH	Flush DMA FIFO. The EMDMA_CTL.DFLSH bit flushes the DMA FIFO. The buffer is only flushed if this bit is set. It can be set with the enable bit. Setting this bit also clears the EMDMA_CTL.DFS bit.
4 (R/W)	CBEN	Circular Buffering Enable. The EMDMA_CTL.CBEN bit enables circular buffering. Circular buffering can be used with normal DMA as well, if circular buffering is enabled with chaining for normal DMA then EIEP and EBEP should be part of the TCB.
		0 Disables circular buffering with delay line DMA
		1 Enables circular buffering with delay line DMA

Table 28-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	DLEN	Enable Delay Line DMA. The EMDMA_CTL.DLEN bit enables delay line DMA. This bit is applicable only if the EMDMA_CTL.CHEN bit =1.
		0 Delay-line DMA disabled
		1 Delay-line DMA enabled
2 (R/W)	CHEN	Enable Chaining. The EMDMA_CTL.CHEN bit enables DMA chaining.
		0 Chaining disabled
		1 Chaining enabled
1 (R/W)	TRAN	DMA Direction. The EMDMA_CTL.TRAN bit determines the DMA data direction. Note: If delay line DMA is enabled then this bit does not have any effect. For delay line DMA, transfer direction depends on the state of delay line transfers. For Internal-Internal or External-External DMA this bit has to be set.
		0 Write through Channel 0 (Channel 1 reads)
		1 Read from Channel 0(Channel 1 writes)
0 (R/W)	EN	DMA Enable. The EMDMA_CTL.EN bit enables DMA.
		0 Disable DMA
		1 Enable DMA

Internal Index Register

The `EMDMA_INDX0` register contains the start address of the buffer for Channel 0 DMA. Note: For delay line DMA the `EMDMA_INDX0` register serves as the delay line write index which is the start address of the channel 0 DMA buffer for the channel 1 write data.

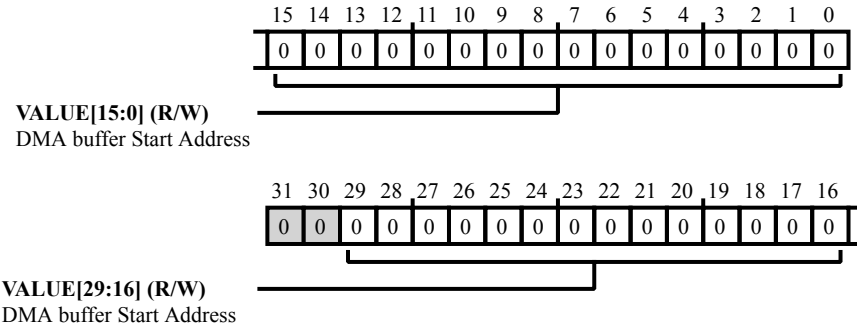


Figure 28-16: EMDMA_INDX0 Register Diagram

Table 28-20: EMDMA_INDX0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	DMA buffer Start Address. The <code>EMDMA_INDX0.VALUE</code> bit field is written with the 30 MSBs of the Word Aligned Byte addresses.

External Index Register

The `EMDMA_IND1` register contains the start address of the buffer for channel 1 DMA.

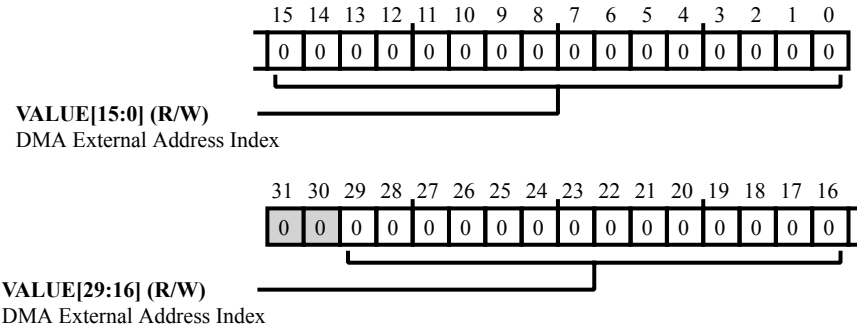


Figure 28-17: EMDMA_IND1 Register Diagram

Table 28-21: EMDMA_IND1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	DMA External Address Index. The <code>EMDMA_IND1.VALUE</code> bit field is the DMA external address index.

Internal Modifier Register

The `EMDMA_MOD0` register contains the channel 0 DMA address modifier.

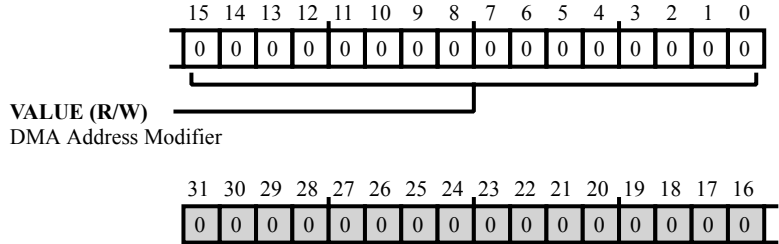


Figure 28-18: EMDMA_MOD0 Register Diagram

Table 28-22: EMDMA_MOD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	DMA Address Modifier. The <code>EMDMA_MOD0.VALUE</code> bit field is the DMA address modifier.

External Modifier Register

The `EMDMA_MOD1` register contains the external (channel 1 DMA) address modifier.

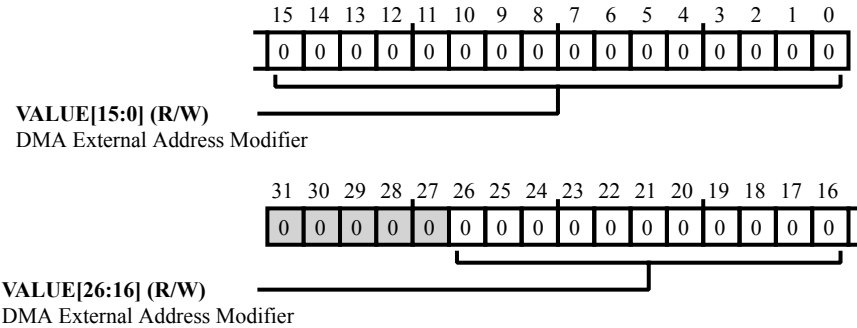


Figure 28-19: EMDMA_MOD1 Register Diagram

Table 28-23: EMDMA_MOD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26:0 (R/W)	VALUE	DMA External Address Modifier. The <code>EMDMA_MOD1.VALUE</code> bit field is the DMA external address modifier.

Delay Line Tap Count Register

The `EMDMA_TCNT` register is the tap count register for Delay Line DMA. This register holds the length of the tap list (the number of taps). The total number of words read from the delay line is equal to the `EMDMA_TCNT` (tap count) multiplied by the `EMDMA_CNT1` (read block size).

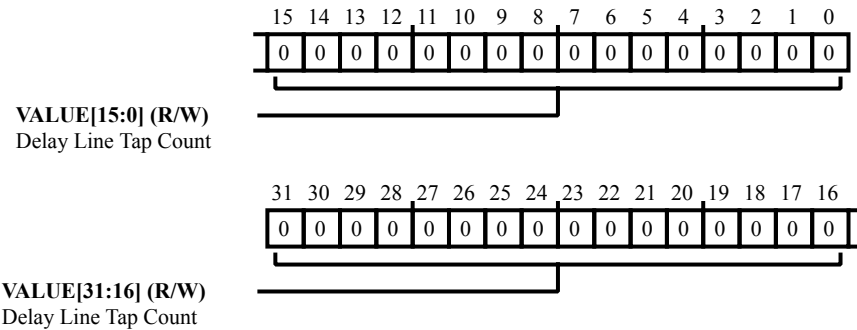


Figure 28-20: EMDMA_TCNT Register Diagram

Table 28-24: EMDMA_TCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Line Tap Count. The <code>EMDMA_TCNT.VALUE</code> bit field is the tap count register for Delay Line DMA.

Tap List Pointer Register

The `EMDMA_TPTR` register holds the address of an array in memory which holds offsets to be used when accessing a Delay-line in system memory. The offset represents the first address of each read block. Note that the lower 30-bits of this register are to be written with 30 MSBs of the Word Aligned Byte address of the Array.

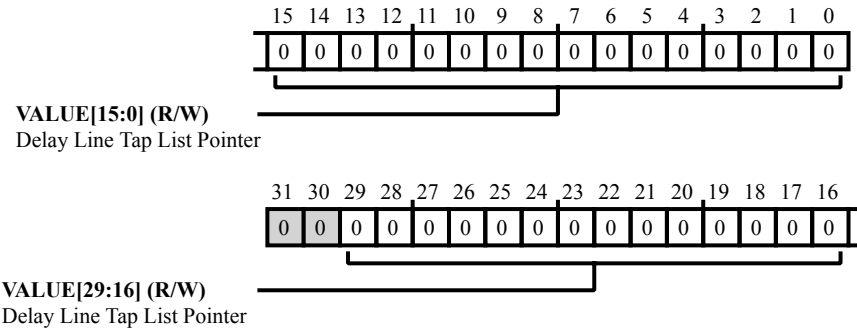


Figure 28-21: EMDMA_TPTR Register Diagram

Table 28-25: EMDMA_TPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Delay Line Tap List Pointer. The <code>EMDMA_TPTR.VALUE</code> bit field contains the offsets to be used when accessing a Delay-line in system memory.

29 Cyclic Redundancy Check (CRC)

The CRC peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error.

The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by any qualified master of the CRC peripheral bus.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the look-up table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

NOTE: CRC is supported by MDMA0 and MDMA1 channels only.

CRC Features

The CRC peripheral supports a number of key features.

- Memory scan modes for memory verification
- Memory transfer modes for on-the-fly CRC calculations while transferring data from one memory to another
- A programmable 32-bit CRC polynomial with automatic LUT generation
- Data mirroring options

The CRC module also includes the following features.

- CRC checksum computation and comparison modes
- 32-bit programmable CRC polynomial with bit reverse option
- Automatic look-up table (LUT) generation

- Data mirroring options for endian and reflected polynomial cases
- Automatic clear and preset of results
- Fault and error interrupt reporting
- DMA and MMR based operation

Because the CRC module is closely tied to memory-to-memory DMA (MDMA) channel pairs, the use cases include the following features.

- Memory scan mode with CRC compute or compare
- Memory transfer mode with CRC compute or compare
- Memory fill operation with 32-bit data patterns
- Memory verify operation
- MMR write access to FIFO of destination DMA
- MMR read access to FIFO of source DMA
- Profiting from advanced DMA features, like descriptor mode and bandwidth control or monitor

CRC Functional Description

The CRC peripheral supports a number of modes of operation that allow for the initialization and verification of regions of memory. The peripheral supports efficient memory-fill and verification operations on regions of memory with or against a constant value. These modes of operation do not require the CRC engine to calculate a signature. Other modes of operation allow for the calculation of CRC signature and verification for a memory region. The modes allow for on-the-fly CRC calculation when performing memory-to-memory DMA transfers from one memory region to another.

To minimize the need for core accesses, the peripheral interfaces with one or more (depending on processor features) memory-to-memory DMA (MDMA) channels. This connectivity permits flexible configuration, in which data can be written-to or read-from the peripheral using DMA transactions, core transactions, or a combination of both.

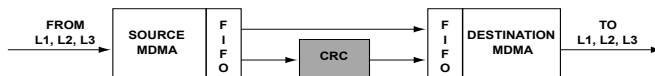


Figure 29-1: Memory Flow

ADSP-2156x CRC Register List

The Cyclic Redundancy Check (CRC) unit includes the data comparison, polynomial operation, and look up table generation features needed for CRC operation. The CRC provides CRC protection as specified by many functional safety requirements. This unit facilitates the system software's ability to periodically check the correctness of the code/data available in memory. A set of registers govern CRC operations. For more information on CRC functionality, see the CRC register descriptions.

Table 29-1: ADSP-2156x CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

ADSP-2156x CRC Interrupt List

Table 29-2: ADSP-2156x CRC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
136	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
137	CRC1_DCNTEXP	CRC1 Datacount expiration	Level	
138	CRC0_ERR	CRC0 Error	Level	
139	CRC1_ERR	CRC1 Error	Level	

CRC Definitions

To make the best use of the CRC, it is useful to understand the following terms.

CRC

Acronym for Cyclic Redundancy Check. An error detection code that can detect changes within a block of data.

CRC Polynomial

The 32-bit polynomial used by the CRC engine to generate the look-up table required for the CRC implementation

LUT

Acronym for the Look-up Table. The look-up table is automatically generated from the supplied 32-bit CRC polynomial.

DMA

Acronym for Direct Memory Access. Used to describe a data transfer that takes place through a DMA channel allowing data distribution around a system without intervention from the core.

MDMA

Acronym for Memory-To-Memory DMA transfer that often requires the use of two DMA channels to transfer data from one memory region to another memory region. One DMA channel is configured as a source channel and the second as a destination channel.

CRC Block Diagram

The *CRC Block Diagram* shows the functional block diagram of the CRC. The following sections describe the blocks.

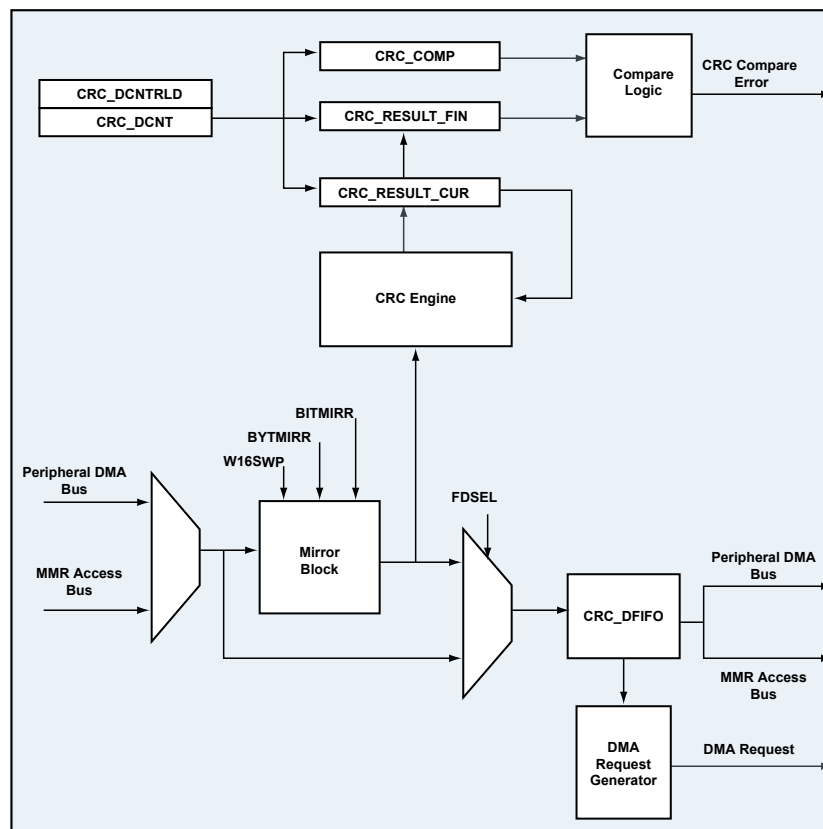


Figure 29-2: CRC Block Diagram

Peripheral DMA Bus

The CRC peripheral provides both an incoming and outgoing datapath to the peripheral DMA bus. The MDMA source channel is interfaced to the incoming datapath providing data to the CRC peripheral. For memory transfer and data fill modes, the CRC uses the MDMA destination channel to either output the data from the CRC FIFO or use the data for the fill operation.

MMR Access Bus

The core uses the MMR access bus to access all the memory-mapped registers of the peripheral for configuration, status, and debug purposes. The core can also use the MMR access bus to feed data to the CRC peripheral or read data from the FIFO of the CRC peripheral. The CRC operation is an alternative to the DMA channel operation to read data from the FIFO.

Data received by MMR writes can transfer to destination DMA. Similarly, data received by source DMA can be output through the MMR interface. Optionally, intermediate results can be made available to the MMR interface.

Mirror Block

The mirror block individually controls bit-reversing of the polynomial, the computation results, and the expected result. Bit mirroring, byte mirroring, word swapping, and any combination of these operations can control endian and the reflection of processed data.

Data FIFO

The CRC data FIFO is a 32-bit-wide 4-entry FIFO. The FIFO is accessible to both the peripheral DMA bus and the MMR access bus. The FIFO status is accessible from the [CRC_STAT](#) register.

DMA Request Generator

The DMA request generator is responsible for granting incoming DMA requests from the source DMA channel and issuing outgoing DMA requests to the destination DMA channel.

CRC Engine

The CRC engine is a 32-bit CRC engine that implements the reduced table look-up scheme. The CRC engine provides support for a user-programmable 32-bit polynomial that the CRC uses to load the look-up table parameters required for the CRC calculation. The CRC engine is a single cycle implementation operating on 32 bits of data per cycle.

Compare Logic

The compare logic takes the final CRC signature and compares it to the expected CRC signature, generating a CRC compare error when the signatures do not match.

CRC Architectural Concepts

A 32-bit polynomial is required before calculation of the CRC signature can occur. The CRC uses the polynomial to generate the contents of an internal look-up table that the reduced table look-up implementation requires. The look-up table is automatically generated when the polynomial is written. It must be initialized prior to any operation that requires the use of the CRC engine.

The mirror block logic can manipulate the data presented to the CRC engine before the CRC uses the data in the calculation of the CRC signature. The data mirror operation is configurable to allow for bit reversing, byte reversing, and 16-bit word swapping operations on the incoming data. For memory transfer compute-and-compare operations, programs can configure the peripheral to output the data in the same form in which it was received. Or, the operation can output the mirrored data in the same manner that it is presented to the CRC engine.

While the CRC peripheral is in operation, the status of the FIFO is continually updated and reflected in the `CRC_STAT` register. The FIFO status is required for core-based accesses to the CRC peripheral. The status indicates when:

- The CRC peripheral can receive data
- Data is available for reading from the FIFO
- The result of the `CRC_RESULT_CUR` register has been updated

The status of the `CRC_RESULT_CUR` register indicates that the current CRC calculation has completed and the result is available.

Look-up Table

The look-up table consists of a set of sixteen 32-bit registers that hardware populates automatically when a write access takes place to the `CRC_POLY` register. 16 clock cycles are required to generate all 16 look-up table entries. The status of the process for generating the look-up table is reflected in `CRC_STAT.LUTDONE` allowing for software to poll on the completion of the event or for generation of an interrupt.

NOTE: Hardware must populate the look-up table before any operation using the CRC peripheral can take place, even if the operation does not use the CRC engine. The peripheral does not issue any data requests until the table generation process is complete. In addition, the `CRC_STAT.IBR` field, that indicates the input buffer status as required for core-based transfers, is only valid upon completion of the process for generating the look-up table.

Data Mirroring

The data mirror block can be configured to manipulate the incoming data before the data passes to the CRC engine and, optionally, to the FIFO. This configuration allows the peripheral to handle various forms of endianness and to function with reflected polynomials.

There are three configuration bits that control the data mirroring process: `CRC_CTL.BITMIRR`, `CRC_CTL.BYTMIRR`, and `CRC_CTL.W16SWP`. The *Data Mirroring Options* table details how these options affect the incoming data and the output generated by the mirror block.

Table 29-3: Data Mirroring Options

W16SWP	BYTMIRR	BITMIRR	Output Data
0	0	0	Dout[31:0] = Din[31:0]
0	0	1	Dout[31:0] = Din[24:31],Din[16:23],Din[8:15],Din[0:7]
0	1	0	Dout[31:0] = Din[7:0],Din[15:8],Din[23:16],Din[31:24]
0	1	1	Dout[31:0] = Din[0:7],Din[8:15],Din[16:23],Din[24:31]
1	0	0	Dout[31:0] = Din[15:0], D[31:16]
1	0	1	Dout[31:0] = Din[8:15],Din[0:7], Din[24:31],Din[16:23]
1	1	0	Dout[31:0] = Din[23:16],Din[31:24], Din[7:0],Din[15:8]
1	1	1	Dout[31:0] = Din[16:23],Din[24:31], Din[0:7],Din[8:15]

When the CRC is configured to operate in the memory transfer compute-and-compare mode, the bit-reversed output data can be written to the FIFO. This feature is controlled through the `CRC_CTL.FDSEL` field.

In addition to providing bit swapping and mirror options to the incoming data, the CRC peripheral also supports bit mirroring on the following registers.

- `CRC_RESULT_CUR` and `CRC_RESULT_FIN`, controlled through the `CRC_CTL.RSLTMIRR` field. When mirroring is enabled, the values to be written to these registers are fully bit-reversed before the write operation occurs.
- `CRC_POLY`, controlled through the `CRC_CTL.POLYMIRR` field. When mirroring is enabled, the 32-bit polynomial is fully bit-reversed before the write operation to the register occurs.
- `CRC_COMP`, controlled through the `CRC_CTL.CMPMIRR` field. When mirroring is enabled, the contents to be loaded to this register are fully bit-reversed before the write operation occurs.

FIFO Status and Data Requests

The CRC peripheral provides indication of the input and output buffer status through `CRC_STAT.IBR` and `CRC_STAT.OBR` respectively. For core-based operations, software must monitor these status fields prior to writing to or reading from the CRC FIFO. No write to the CRC FIFO can occur while `CRC_STAT.IBR` indicates that the buffer is not ready to accept data. Similarly, the CRC FIFO cannot be read until `CRC_STAT.OBR` indicates that data is available.

The memory scan modes of operation only require the monitoring of the input buffer status. The memory transfer, compute-and-compare mode uses both input and output buffer status. If the current result of the CRC computation is required, then software must verify that the current operation has completed and that the intermediate result is ready. The `CRC_STAT.IRR` indicates the status.

NOTE: The memory transfer fill mode of operation requires the use of a DMA channel. The CRC does not support core reads from the CRC FIFO for this mode of operation.

Memory transfer, compute-and-compare mode uses burst transactions to make the most efficient use of the available resources. In this mode, when the FIFO is initially empty and the peripheral is enabled, the `CRC_STAT.IBR` bit indicates that the CRC is ready to accept data. The peripheral generates data requests to the source DMA channel (if the CRC uses DMA). While the number of words remaining in the `CRC_DCNT` register is greater than the FIFO depth, the peripheral issues data requests or accepts incoming data in bursts. The peripheral continues until the CRC FIFO becomes full.

Once full, the `CRC_STAT.IBR` and `CRC_STAT.OBR` bits are updated, and then the CRC issues outgoing data requests. Only when the FIFO is empty can the peripheral accept further incoming data, and the `CRC_STAT.IBR` and `CRC_STAT.OBR` bits are updated once again.

Once `CRC_DCNT` is decremented such that the number of words waiting for processing is less than the number required to fill the FIFO, the burst mode of operation is disabled. Incoming data is accepted as long as the FIFO is not full. Outgoing data is available as long the FIFO is not empty. Therefore, there are no restrictions requiring the word count to be a multiple of the FIFO depth.

All other CRC modes of operation indicate that incoming data can be accepted as long as the FIFO is not full. Outgoing data is available as long as the FIFO is not empty.

The `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit configurations also influence how the CRC generates data requests and status bits. The following list describes the bits.

- The `CRC_CTL.OBRSTALL` bit can be configured such that the CRC peripheral stalls as soon as there is output data available in the FIFO. Use this mode of operation only in memory transfer, compute-and-compare mode. This mode results in the processing of one 32-bit word at a time. The peripheral does not request or accept incoming data until the current value being processed is read from the peripheral.
- The `CRC_CTL.IRRSTALL` bit can be configured so that the CRC peripheral stalls all further incoming data requests until the `CRC_RESULT_CUR` register is read after being updated. Use this mode of operation for CRC signature generation. It is not applicable to memory transfer data-fill mode or memory scan data-verify mode of operation.

CRC Operating Modes

The following sections describe the various operating modes of the CRC interface.

Data Transfer Modes

The CRC peripheral supports two main categories of operation involving data transfers:

- Memory scan mode
- Memory transfer mode

Memory scan modes are read-only operations that allow the contents of memory to be read into the peripheral and verified for correctness. There are two forms of memory scan mode:

- CRC compute-and-compare performs a CRC calculation on data presented to the peripheral and compares the CRC result with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data verify compares each 32-bit data word presented to the CRC peripheral to a pre-loaded 32-bit value and generates an error when the data differs.

Both of these modes of operation require, at the most, a single DMA channel to read the data from memory into the peripheral. No data is forwarded to the data output or destination DMA. The CRC can also use core-driven transfers for either of these modes of operation.

The memory transfer modes involve memory write or memory read-and-write operations allowing for memory to be initialized or transferred from one region of memory to another. There are two forms of memory transfer mode:

- CRC compute-and-compare performs a full data transfer from one memory region to another memory region. The CRC generates a signature on the data presented and compares it with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data fill initializes a region of memory with a pre-loaded 32-bit constant value.

The CRC compute-and-compare mode of operation requires both incoming and outgoing data channels. The operation occurs either using DMA channels, using core-driven write or read operations to and from the FIFO or using a combination of both. The data fill mode of operation requires only a memory write DMA destination channel—this mode does not support core driven operations.

Memory Scan Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the `CRC_CTL.OPMODE` field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the `CRC_DCNT` register. The accumulated result of the CRC operation is contained in the `CRC_RESULT_CUR` register. As the CRC engine processes each 32-bit word, the `CRC_DCNT` register is decremented and `CRC_RESULT_CUR` is updated.

Once `CRC_DCNT` decrements to zero, the contents of the `CRC_RESULT_CUR` register are copied to `CRC_RESULT_FIN` and `CRC_STAT.DCNTEXP` is updated accordingly. The CRC uses the `CRC_COMP` register to store the expected result of the operation. After the CRC calculation, `CRC_COMP` is compared with `CRC_RESULT_FIN` and `CRC_STAT.CMPERR` is updated to reflect the status of the compare operation. `CRC_STAT.CMPERR` must be cleared before the next CRC operation is performed.

The CRC peripheral also contains the `CRC_DCNTRLD` register. The CRC uses this register to reload `CRC_DCNT` upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRZ`. This configuration provides a way to reset `CRC_RESULT_CUR` to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Scan Data Verify

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the `CRC_CTL.OPMODE` field. Each 32-bit word of the data stream is compared with a constant value that is stored in the `CRC_COMP` register. The `CRC_DCNT` register contains the number of words for comparison. The `CRC_DCNT` register is decremented upon receiving a new 32-bit word from the data stream. If the compare operation fails, the `CRC_STAT.CMPERR` bit is updated and the contents of `CRC_DCNT` are captured in the `CRC_DCNTCAP` register. This information can be used to identify the location in the data stream where the error occurred. Clear the `CRC_STAT.CMPERR` field to reenables capturing of further errors.

Once `CRC_DCNT` decrements to zero, `CRC_STAT.DCNTEXP` is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the `CRC_CTL.OPMODE` field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the `CRC_DCNT` register. The accumulated result of the CRC operation is contained in the `CRC_RESULT_CUR` register. As the CRC engine processes each 32-bit word, the `CRC_DCNT` register is decremented and `CRC_RESULT_CUR` is updated.

Once `CRC_DCNT` decrements to zero, the contents of the `CRC_RESULT_CUR` register are copied to `CRC_RESULT_FIN` and `CRC_STAT.DCNTEXP` is updated accordingly. The CRC uses the `CRC_COMP` register to store the expected result of the operation. Upon completion of the CRC calculation, `CRC_COMP` is compared with `CRC_RESULT_FIN` and `CRC_STAT.CMPERR` is updated to reflect the status of the compare operation. Clear `CRC_STAT.CMPERR` before the next CRC operation is performed.

The CRC peripheral also contains `CRC_DCNTRLD` register. The CRC uses this register to reload `CRC_DCNT` upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRF`. This configuration provides a means to reset `CRC_RESULT_CUR` to `0x00000000`, `0xFFFFFFFF` or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Data Fill Mode

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the `CRC_CTL.OPMODE` field. The `CRC_FILLVAL` register is written with a 32-bit value. The CRC uses this value to initialize a block memory through the memory-to-memory DMA destination channel. When the CRC peripheral and the DMA destination channel are enabled, the contents of the `CRC_FILLVAL` register is written to the DMA channel to initialize the memory region. The `CRC_DCNT` register contains the number of words for the write operation.

Once `CRC_DCNT` decrements to zero, `CRC_STAT.DCNTEXP` is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the data expiration event to generate an interrupt.

CRC Event Control

The CRC peripheral can enable certain CRC status operations to generate an interrupt event to the system event controller. There, a CRC error can be qualified as a system fault.

Interrupt Signals

The CRC peripheral can generate two interrupt requests that are enabled as events to SEC1 for fault operations. One is a CRC status interrupt and the other is a CRC error interrupt.

The `CRC_STAT.CMPERR` status bit can be configured as an interrupt and is signaled through the CRC error interrupt signal. The `CRC_STAT.CMPERR` status field is set whenever the CRC peripheral performs a compare operation that fails. This status can be the result of a failed memory scan data-verify operation that compares the contents of a memory range with a constant 32-bit value. Or, it can be the result of a CRC signature calculated for a memory region that does not match the expected pre-programmed result for a memory-compare operation.

The `CRC_STAT.DCNTEXP` status bit is set when the `CRC_DCNT` register has decremented to zero. The status indicates that the CRC peripheral has now processed all the data requested for the current CRC operation. The CRC can also use this signal to generate an interrupt. The interrupt is signaled on the CRC status interrupt signal.

Both these status bits can be configured to generate an interrupt through the `CRC_INEN` register. The `CRC_INEN` register also has bit set, `CRC_INEN_SET`, and bit clear `CRC_INEN_CLR` equivalent registers that the CRC uses for the enabling and disabling of these interrupt sources.

The `CRC_STAT` register has two write one to clear (W1C) fields for clearing the two interrupt sources.

NOTE: Disabling the CRC peripheral through the `CRC_CTL.BLKEN` bit does not result in the clearing of interrupt sources. Clear the interrupt sources using a W1C operation to the `CRC_STAT` register.

CRC Programming Model

It is important to note the following restrictions when using the CRC peripheral with the DMA channels:

1. When enabling the CRC peripheral and the DMA channels, enable the CRC peripheral prior to enabling the DMA channels.
2. When disabling the CRC peripheral and the DMA channels, disable the DMA channels prior to disabling the CRC peripheral.

CRC Mode Configuration

Describes a number of tasks showing the various operation modes of the CRC peripheral.

- [Look-up Table Generation](#)

- Core Driven Memory Scan Compute-and-Compare Mode
- DMA Driven Memory Scan Compute-and-Compare Mode
- Core Driven Memory Scan Data Verify Mode
- DMA Driven Memory Scan Data Verify Mode
- Core Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Data Fill Mode

Look-up Table Generation

Describes the steps required to initialize the CRC peripheral LUT.

1. Write the 32-bit CRC polynomial of choice to the `CRC_POLY` register.

ADDITIONAL INFORMATION: This operation results in the CRC peripheral starting the LUT initialization process. The `CRC_STAT.LUTDONE` bit is updated to reflect the operation is in progress.

2. Poll the `CRC_STAT.LUTDONE` bit until the status bit indicates that the operation is completed.

The CRC peripheral has completed initialization of all the LUT registers and is now ready for data operations. The `CRC_STAT.LUTDONE` bit remains in the current state until the `CRC_POLY` register is written again, or the peripheral or processor are reset.

Core Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using core transactions. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling the `CRC_CTL` register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per `CRC_CTL.BLKEN`

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute-and-compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

- Disable the `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit options for this task example.
- Configure all mirroring and bit reversal options.
- Configure CRC auto-clear options.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral.

- a. While `CRC_STAT.IBR` bit indicates that the input buffer is ready, write the `CRC_DFIFO` register with 32-bit data.

ADDITIONAL INFORMATION: Repeat this step until all data has been written.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write to the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear of these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed. The final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled.

Clear any WIC CRC status bits before performing more CRC operations.

DMA Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using DMA transactions. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling `CRC_CTL`.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit.

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final operation.

5. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

- Disable the `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit options for this task example.
- Configure all mirroring and bit reversal options.
- Configure all CRC auto clear options.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed and the final result indicated is through `CRC_STAT.CMPERR` and the corresponding interrupt, when enabled.

Clear any WIC CRC status bits before performing a further CRC operation. Clear any WIC status bits of the memory-to-memory source DMA channel before the next CRC operation.

Core Driven Memory Scan Data Verify Mode

Reads a region of memory using core transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per `CRC_CTL.BLKEN`

The interrupt service routine for the compare error interrupt reads and stores the contents of `CRC_DCNTCAP` register to a buffer before clearing the compare error interrupt.

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan data verify mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

6. Write memory region data to the CRC peripheral.
 - a. Poll the `CRC_STAT.IBR` bit until input buffer is ready.
 - b. Write the `CRC_DFIFO` register with 32-bit data.

ADDITIONAL INFORMATION: Repeat these two steps until the entire memory region has been written to the CRC peripheral.

7. Poll the `CRC_INEN_SET.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the `CRC_DCNTCAP` register upon a compare error has any new entries.

ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write to the `CRC_STAT` to clear both the `CRC_INEN_SET.DCNTEXP` and `CRC_INEN.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, the clear these status bits within the interrupt handlers for the respective interrupts.

The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the `CRC_INEN.CMPERR` bit and the corresponding interrupt, when enabled. Each comparison error is traceable due to the logging of `CRC_DCNTCAP` from within the compare error interrupt handler.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Scan Data Verify Mode

The memory scan data verify mode reads a region of memory using DMA transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

The interrupt service routine for the compare error interrupt reads and stores the contents of the `CRC_DCNTCAP` register to a buffer before clearing the compare error interrupt.

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: The CRC module uses this register to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan data verify mode and `CRC_CTL.BLKEN` configured to enable the CRC peripheral.

The CRC peripheral is now enabled and ready for the core or DMA channel to write the data.

6. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

7. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the `CRC_DCNTCAP` register upon a compare error has any new entries.

ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled. Each comparison error is traceable due to the logging of the `CRC_DCNTCAP` register from within the compare error interrupt handler.

Clear any WIC CRC status bits and DMA status bits before performing a further CRC operation.

Core Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using core transactions while copying the contents to another memory region. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling the `CRC_CTL` register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of the current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute-and-compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.
 - a. Disable the `CRC_CTL.OBRSTALL` bit and the `CRC_CTL.IRRSTALL` bit options for this task example.
 - b. Configure all mirroring and bit reversal options.
 - c. Configure CRC auto clear options

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral and read it back to the new destination.
 - a. While the `CRC_STAT.IBR` bit indicates that the input buffer is ready, write the `CRC_DFIFO` register with 32-bit data.
 - b. While the `CRC_STAT.OBR` bit indicates that the output buffer is ready, read the `CRC_DFIFO` register and store data to new destination.

ADDITIONAL INFORMATION: Repeat these two steps until all required data has been processed through the CRC peripheral and copied to the new destination.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The memory region has been copied to a new location and an integrity check of the memory through the expected CRC signature has also completed. The final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using DMA transactions. The memory region is also copied to another memory region using memory-to-memory DMA transfers. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling `CRC_CTL`.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` register.

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute compare mode and `CRC_CTL.BLKEN` configured to enable the CRC peripheral.

- a. Disable the `CRC_CTL.OBRSTALL` and the `CRC_CTL.IRRSTALL` bit options for this task example.
- b. Configure all mirroring and bit reversal options
- c. Configure CRC auto clear options

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode and destination DMA channel for memory write STOP mode.

ADDITIONAL INFORMATION: This step starts the data transfer from one memory region to another through the memory-to-memory DMA channels and the CRC peripheral.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and the `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The integrity check of the memory through the expected CRC signature has completed and the final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled. The memory region has also been copied to its final destination.

Clear any W1C CRC status bits before performing a further CRC operation. Also, clear any W1C status bits of the memory-to-memory source and destination DMA channels before the next CRC operation.

DMA Driven Memory Transfer Data Fill Mode

This mode initializes a region of memory to a constant 32-bit value using DMA transactions.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_FILLVAL` register.

ADDITIONAL INFORMATION: This register contains the 32-bit constant that the CRC module uses to fill the memory region.

4. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of block completion. Configure these interrupts as required. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory transfer fill mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

The CRC peripheral is now enabled and is ready for the DMA channel to write data.

6. Configure and enable the memory-to-memory destination DMA channel for memory write STOP mode.

ADDITIONAL INFORMATION: This step starts the data transfer taking the constant 32-bit value from the CRC peripheral and writing the data to the DMA channel.

7. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

8. Write the `CRC_STAT` register to clear the `CRC_STAT.DCNTEXP` bit.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear this status bit within the interrupt handlers for the respective interrupts.

The CRC memory transfer fill operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

The memory region is now filled with the constant data and the CRC peripheral is ready to be configured for a new operation.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

ADSP-2156x CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 29-4: ADSP-2156x CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

Data Compare Register

The `CRC_COMP` register contains the value corresponding to the expected CRC result or signature for the current data stream. At the end of the operation, the content of this register is used to compare against the result produced by the CRC operation. In data verify mode, each incoming data value is compared with the content of this register.

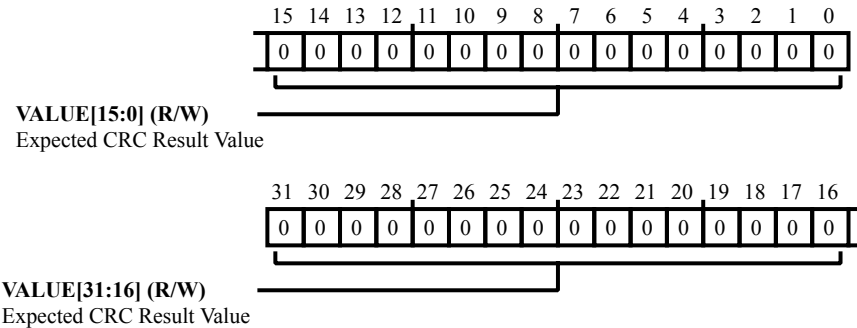


Figure 29-3: CRC_COMP Register Diagram

Table 29-5: CRC_COMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Expected CRC Result Value. The <code>CRC_COMP.VALUE</code> bit field contains the value corresponding to the expected CRC result or signature for the current data stream.

Control Register

The `CRC_CTL` register configures the operation modes and settings for the CRC.

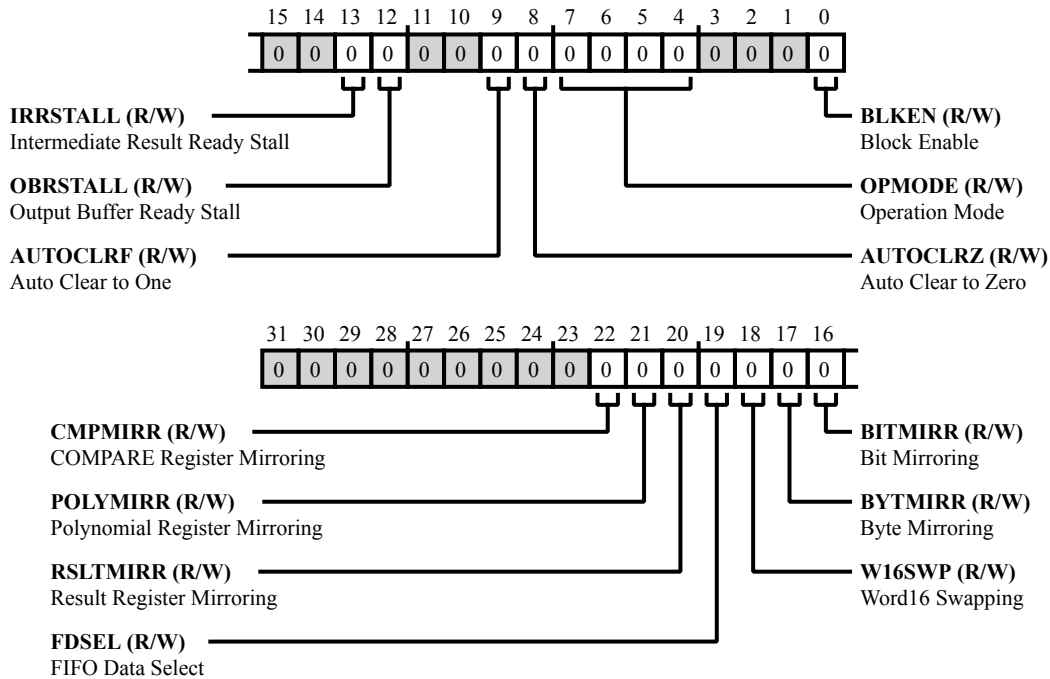


Figure 29-4: CRC_CTL Register Diagram

Table 29-6: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	CMPMIRR	COMPARE Register Mirroring. The <code>CRC_CTL.CMPMIRR</code> bit enables data mirroring for the <code>CRC_COMP</code> compare register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for comparison with the <code>CRC_RESULT_FIN</code> register.
		0 Disable compare mirroring
		1 Enable compare mirroring
21 (R/W)	POLYMIRR	Polynomial Register Mirroring. The <code>CRC_CTL.POLYMIRR</code> bit enables data mirroring for the <code>CRC_POLY</code> polynomial register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for CRC computations.
		0 Disable polynomial mirroring
		1 Enable polynomial mirroring

Table 29-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	RSLTMIRR	Result Register Mirroring. The <code>CRC_CTL.RSLTMIRR</code> bit enables data mirroring for the <code>CRC_RESULT_CUR</code> and <code>CRC_RESULT_FIN</code> result registers. When enabled, the 32-bit values in these registers are fully bit mirrored (reversed).
		0 Disable result mirroring
		1 Enable result mirroring
19 (R/W)	FDSEL	FIFO Data Select. The <code>CRC_CTL.FDSEL</code> bit selects whether the CRC writes modified or unmodified data to the FIFO in memory transfer mode. If enabled, the data written is affected by the state of the data mirroring selections (<code>CRC_CTL.BITMIRR</code> , <code>CRC_CTL.BYTMIRR</code> , and <code>CRC_CTL.W16SWP</code>) before being written to the FIFO.
		0 Write unmodified data to FIFO
		1 Write modified data to FIFO
18 (R/W)	W16SWP	Word16 Swapping. The <code>CRC_CTL.W16SWP</code> bit enables the CRC's data mirror block to swap the upper and lower 16-bit words within the 32-bit input data, before further processing.
		0 Disable word16 swapping
		1 Enable word16 swapping
17 (R/W)	BYTMIRR	Byte Mirroring. The <code>CRC_CTL.BYTMIRR</code> bit enables the CRC's data mirror block to mirror the bytes within the 32-bit input data, before further processing.
		0 Disable byte mirroring
		1 Enable byte mirroring
16 (R/W)	BITMIRR	Bit Mirroring. The <code>CRC_CTL.BITMIRR</code> bit enables the CRC's data mirror block to mirror the bits within each byte of the 32-bit input data, before further processing.
		0 Disable bit mirroring
		1 Enable bit mirroring
13 (R/W)	IRRSTALL	Intermediate Result Ready Stall. The <code>CRC_CTL.IRRSTALL</code> bit enables stalling the state machine for input data when there is a valid intermediate result to be read in the <code>CRC_RESULT_CUR</code> register. This feature should be used only in CRC computation modes (for example, <code>CRC_CTL.OPMODE = 1</code> or <code>=3</code>).
		0 Do not stall
		1 Stall on IRR

Table 29-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	OBRSTALL	Output Buffer Ready Stall. The CRC_CTL.OBRSTALL bit enables stalling the state machine for input data when there is valid data in the output buffer. This feature should be used only in memory-to-memory transfer modes (for example, CRC_CTL.OPMODE =1).
		0 Do not stall
		1 Stall on OBR
9 (R/W)	AUTOCLRF	Auto Clear to One. The CRC_CTL.AUTOCLRF bit enables auto clear to one when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that the CRC_CTL.AUTOCLRZ bit must be disabled, or the CRC_CTL.AUTOCLRF bit has no effect.
		0 No auto clear
		1 Auto clear
8 (R/W)	AUTOCLRZ	Auto Clear to Zero. The CRC_CTL.AUTOCLRZ bit enables auto clear to zero when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that CRC_CTL.AUTOCLRF must be disabled, or the CRC_CTL.AUTOCLRZ has no effect.
		0 No auto clear
		1 Auto clear
7:4 (R/W)	OPMODE	Operation Mode. The CRC_CTL.OPMODE bit field selects the memory transfer or scan mode.
		0 Reserved
		1 CRC compute/compare memory transfer
		2 Data fill memory transfer
		3 CRC compute/compare memory scan
		4 Data verify memory scan
0 (R/W)	BLKEN	Block Enable. The CRC_CTL.BLKEN bit enables and disables the CRC operation.
		0 Disable
		1 Enable

Data Word Count Register

The `CRC_DCNT` register holds the word count that is used for the CRC operation. On transfer of every 32-bit word, the CRC decrements by 1 the content of this register. When the count decrements to zero, this event triggers a CRC compare action, and the `CRC_DCNT` register is automatically loaded from the `CRC_DCNTRLD` register for the next CRC operation.

Note that the initial value programmed into the `CRC_DCNT` register may be different from what is programmed in the `CRC_DCNTRLD` register.

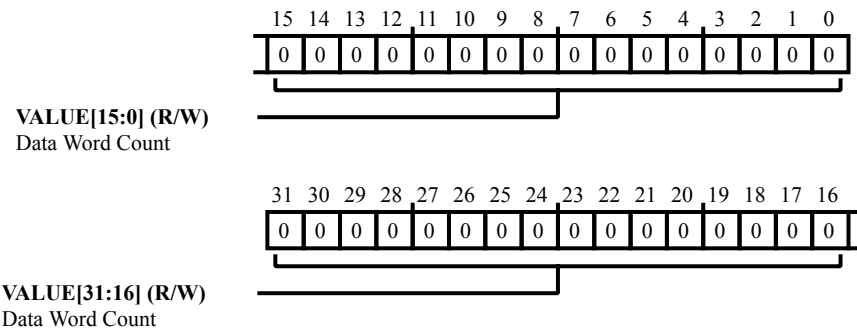


Figure 29-5: CRC_DCNT Register Diagram

Table 29-7: CRC_DCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Word Count. The <code>CRC_DCNT.VALUE</code> bit field holds the word count that is used for the CRC operation.

Data Count Capture Register

The `CRC_DCNTCAP` register captures the `CRC_DCNT` value when a compare operation fails in data verify mode. This capture can be used to track the position of an error in the data stream. The capture operation is enabled only if the `CRC_STAT.CMPERR` bit indicates no compare error. After an error occurs and the data count is captured, no further errors are logged until the `CRC_STAT.CMPERR` bit is cleared. To obtain the position of an error in the data stream, subtract the `CRC_DCNTCAP` register value from the initial `CRC_DCNT`.

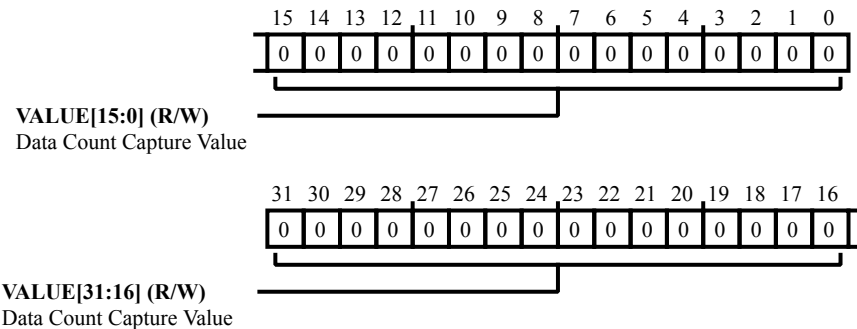


Figure 29-6: CRC_DCNTCAP Register Diagram

Table 29-8: CRC_DCNTCAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Count Capture Value. The <code>CRC_DCNTCAP.VALUE</code> bit field contains the <code>CRC_DCNT</code> value when a compare operation fails in data verify mode.

Data Word Count Reload Register

The `CRC_DCNTRLD` register holds the value that the CRC automatically loads into `CRC_DCNT` when the `CRC_DCNT` decrements to 0. At startup, the value programmed in `CRC_DCNT` and the `CRC_DCNTRLD` register could be different. So, for the first iteration, the CRC operation happens for the count initially programmed in the `CRC_DCNT` register. While for subsequent CRC operations, the count is taken from the `CRC_DCNTRLD` register.

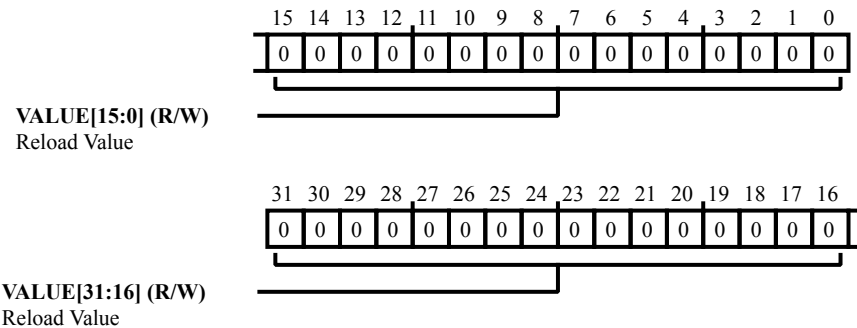


Figure 29-7: CRC_DCNTRLD Register Diagram

Table 29-9: CRC_DCNTRLD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reload Value. The <code>CRC_DCNTRLD.VALUE</code> bit field holds the value that automatically loads into <code>CRC_DCNT</code> when the <code>CRC_DCNT</code> decrements to 0.

Data FIFO Register

In memory transfer mode (non-data fill mode), the data from the DMA or processor core buses is written into the `CRC_DFIFO` on each input data grant (DMA grant or core write). Data is read from this FIFO on each output data grant (DMA grant or core read). FIFO status information is available in the `CRC_STAT` register. Whenever, the FIFO has valid data, output data requests are generated.

Note that in non-memory transfer mode and in data fill mode, the input data does not get written into this FIFO. So, this register should not be read in these modes.

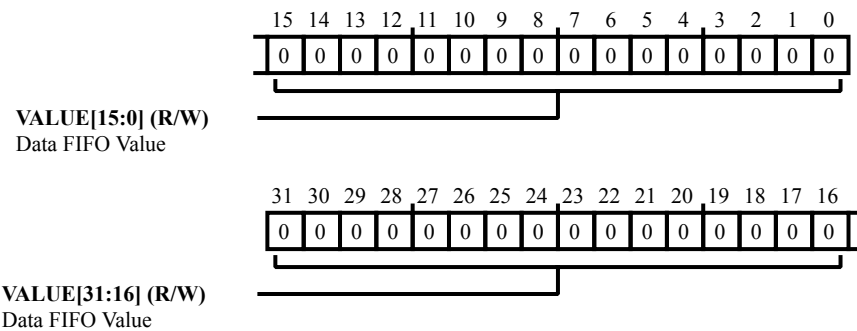


Figure 29-8: CRC_DFIFO Register Diagram

Table 29-10: CRC_DFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data FIFO Value. The <code>CRC_DFIFO.VALUE</code> bit field is the data from the DMA or processor core buses.

Fill Value Register

The `CRC_FILLVAL` register holds the value that the CRC uses for the memory fill operation. In data fill mode, the value programmed in this register is used for the memory fill operation.

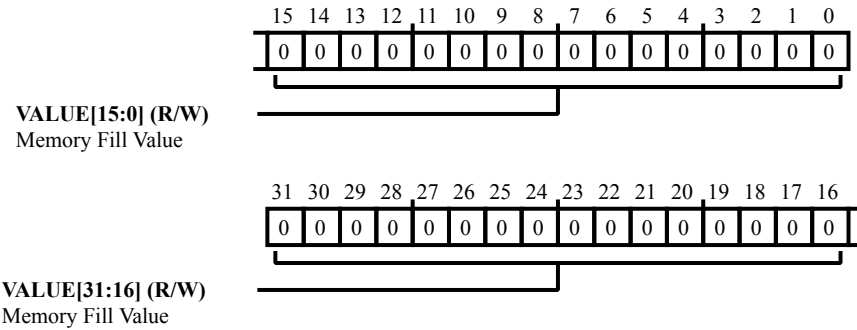


Figure 29-9: CRC_FILLVAL Register Diagram

Table 29-11: CRC_FILLVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Memory Fill Value. The <code>CRC_FILLVAL.VALUE</code> bit field holds the value that the CRC uses for the memory fill operation.

Interrupt Enable Register

The `CRC_INEN` register unmask (enables) or mask (disables) interrupt requests generated in the CRC from going to the processor core.

Note that CRC interrupts are not disabled when the CRC is disabled (`CRC_CTL.BLKEN = 0`).

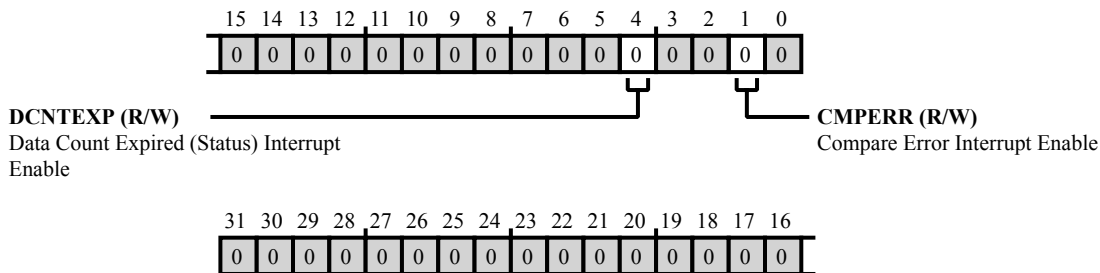


Figure 29-10: CRC_INEN Register Diagram

Table 29-12: CRC_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	DCNTEXP	Data Count Expired (Status) Interrupt Enable. The <code>CRC_INEN.DCNTEXP</code> enables (unmasks) the data count expired (CRC status) interrupt.
		0 Disable (mask) interrupt
		1 Enable (unmask) interrupt
1 (R/W)	CMPERR	Compare Error Interrupt Enable. The <code>CRC_INEN.CMPERR</code> enables (unmasks) the data compare interrupt, which is generated when CRC data comparison fails.
		0 Disable (mask) interrupt
		1 Enable (unmask) interrupt

Interrupt Enable Clear Register

The `CRC_INEN_CLR` register permits clearing individual bits in the `CRC_INEN` register without affecting other bits in the register.

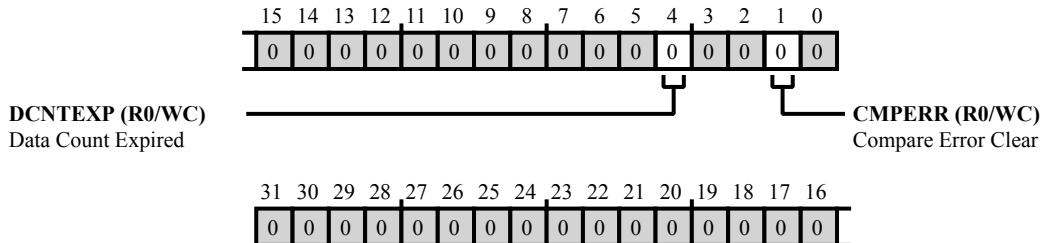


Figure 29-11: CRC_INEN_CLR Register Diagram

Table 29-13: CRC_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R0/WC)	DCNTEXP	Data Count Expired. The <code>CRC_INEN_CLR.DCNTEXP</code> bit clears the data count expired (status) interrupt.
		0 No effect
		1 Clear bit
1 (R0/WC)	CMPERR	Compare Error Clear. The <code>CRC_INEN_CLR.CMPERR</code> bit clears the compare error interrupt.
		0 No effect
		1 Clear bit

Interrupt Enable Set Register

The `CRC_INEN_SET` register permits setting individual bits in the `CRC_INEN` register without affecting other bits in the register.

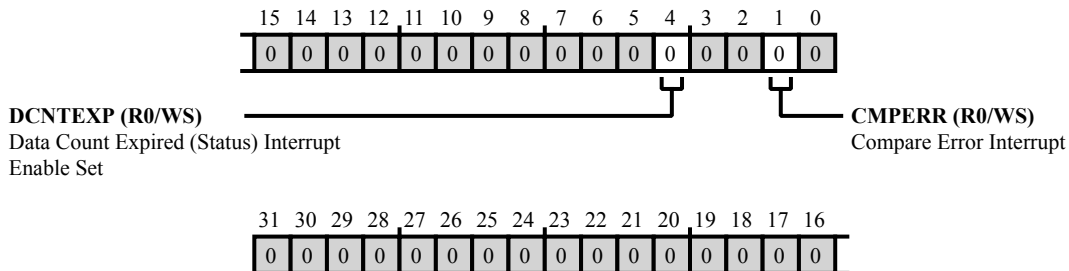


Figure 29-12: `CRC_INEN_SET` Register Diagram

Table 29-14: `CRC_INEN_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R0/WS)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Set. The <code>CRC_INEN_SET.DCNTEXP</code> bit sets the data count expired (status) interrupt.
		0 No effect
		1 Set bit
1 (R0/WS)	CMPERR	Compare Error Interrupt. The <code>CRC_INEN_SET.CMPERR</code> bit sets the compare error interrupt.
		0 No effect
		1 Set bit

Polynomial Register

The `CRC_POLY` register holds a 32-bit polynomial for CRC operations. Bit 31 corresponds to the coefficient of x^{31} of the CRC polynomial, bit 30 corresponds to the coefficient of x^{30} , and so on through bit 0. A coefficient of x^{32} is assumed to be "1" for any polynomial that is selected. Based on the polynomial in the `CRC_POLY` register, the CRC generates a look-up table (LUT), which is used to compute the CRC of the incoming data stream.

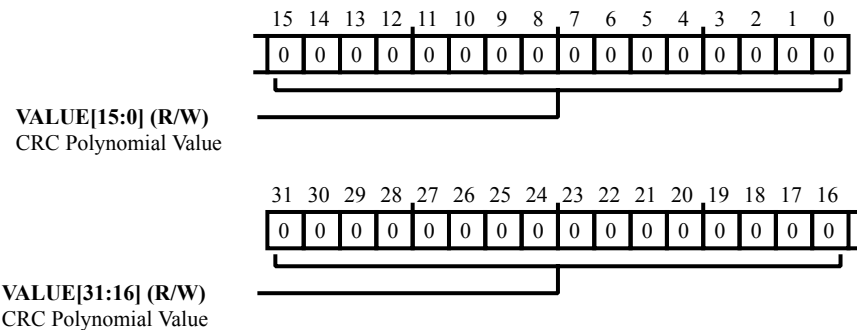


Figure 29-13: CRC_POLY Register Diagram

Table 29-15: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Polynomial Value. The <code>CRC_POLY.VALUE</code> bit field holds the 32-bit polynomial for CRC operations.

CRC Current Result Register

The `CRC_RESULT_CUR` register holds the current or intermediate CRC result. It is updated when new data is written into the CRC. Each time the `CRC_DCNT` expires, the CRC loads the value from this register into the `CRC_RESULT_FIN` register. The `CRC_RESULT_CUR` register may be set to auto clear to zero or auto clear to ones when `CRC_DCNT` expires by configuring the `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRF` bits. Before starting a CRC operation, the `CRC_RESULT_CUR` register should be programmed to the desired value.

Note that this register can be read by the processor core at any time.

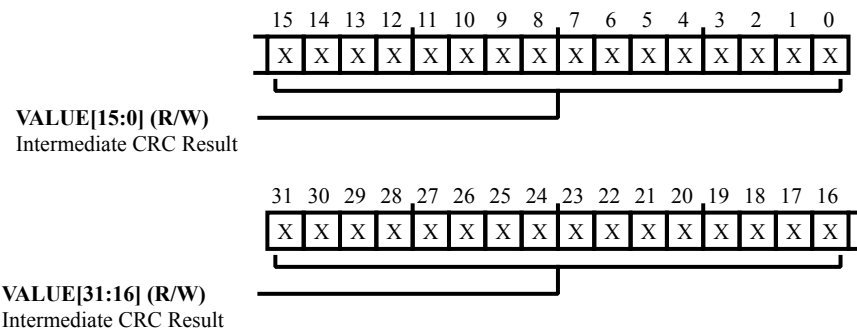


Figure 29-14: `CRC_RESULT_CUR` Register Diagram

Table 29-16: `CRC_RESULT_CUR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Intermediate CRC Result. The <code>CRC_RESULT_CUR.VALUE</code> bit field holds the current or intermediate CRC result.

CRC Final Result Register

The `CRC_RESULT_FIN` register holds the final CRC computed for a data stream. A data stream is a DMA of `CRC_DCNT` number of words into the CRC. When `CRC_DCNT` decrements to zero for each data stream, the CRC loads the `CRC_RESULT_FIN` register with the value from the `CRC_RESULT_CUR` register.

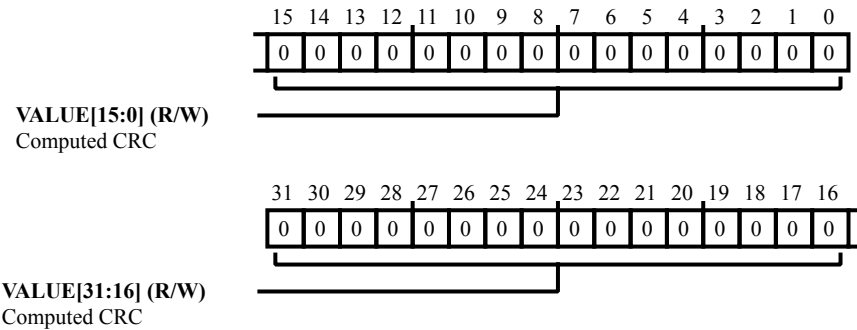


Figure 29-15: CRC_RESULT_FIN Register Diagram

Table 29-17: CRC_RESULT_FIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Computed CRC. The <code>CRC_RESULT_FIN.VALUE</code> bit field holds the final CRC computed for a data stream.

Status Register

The `CRC_STAT` register indicates the status for CRC operations and interrupt generation.

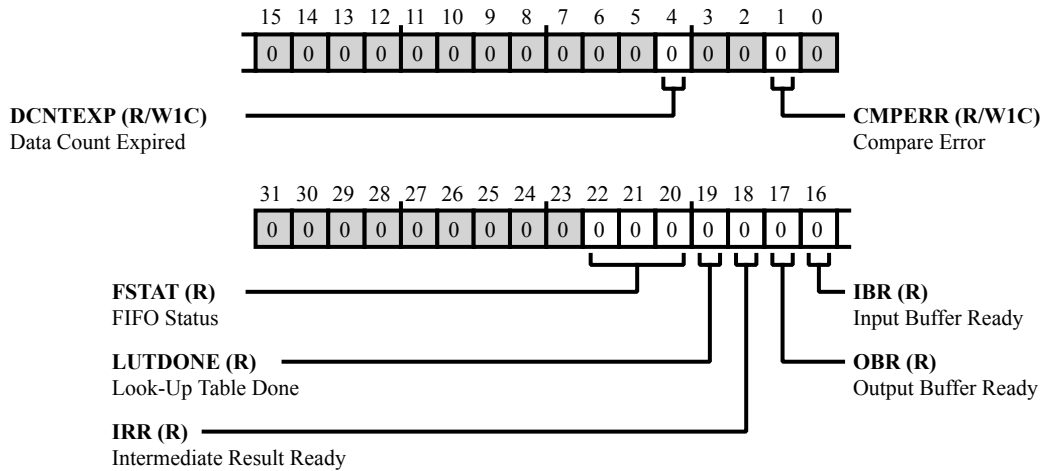


Figure 29-16: CRC_STAT Register Diagram

Table 29-18: CRC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22:20 (R/NW)	FSTAT	FIFO Status. The <code>CRC_STAT.FSTAT</code> indicates the current FIFO status. This field is read-only.
		0 FIFO empty
		1 FIFO has 1 data
		2 FIFO has 2 data
		3 FIFO has 3 data
19 (R/NW)	LUTDONE	Look-Up Table Done. The <code>CRC_STAT.LUTDONE</code> bit indicates that the CRC has generated the look-up table for the current polynomial. This read-only bit is cleared at reset and cleared when the <code>CRC_POLY</code> is written.
		0 No status
		1 LUT generation done

Table 29-18: CRC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IRR	Intermediate Result Ready. The CRC_STAT . IRR bit indicates that the CRC has updated the CRC_RESULT_CUR register with intermediate CRC results for the new data written to the CRC. The processor core should read from the CRC_RESULT_CUR register only after detecting CRC_STAT . IRR =1. This read-only bit is cleared by CRC hardware and is valid when the CRC_CTL . IRRSTALL bit is enabled.
		0 No status
		1 Intermediate results ready
17 (R/NW)	OBR	Output Buffer Ready. The CRC_STAT . OBR bit indicates that the CRC has data ready for the processor core to read. The processor core should read from the CRC only after detecting CRC_STAT . OBR =1. This read-only bit is cleared by CRC hardware.
		0 No status
		1 Output buffer ready
16 (R/NW)	IBR	Input Buffer Ready. The CRC_STAT . IBR bit indicates that the CRC is ready to accept a processor core write. The processor core should write to the input register only after detecting that CRC_STAT . IBR =1. This read-only bit is cleared by CRC hardware.
		0 No status
		1 Input buffer ready
4 (R/W1C)	DCNTEXP	Data Count Expired. The CRC_STAT . DCNTEXP bit indicates that the CRC_DCNT has expired. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL . BLKEN =0). When the CRC sets this bit on CRC_DCNT expiry, the CRC generates the CRC_INEN . DCNTEXP interrupt.
		0 No status
		1 Data counter expired
1 (R/W1C)	CMPERR	Compare Error. The CRC_STAT . CMPERR bit indicates that a CRC mismatch or data mismatch has been detected. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL . BLKEN =0). When the CRC sets this bit on detecting a mismatch, the CRC generates the CRC_INEN . CMPERR interrupt. While this bit is set, the CRC_DCNTCAP register is disabled from capturing the data count values.
		0 No status
		1 Compare error

30 Housekeeping ADC (HADC)

The Housekeeping ADC is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters like voltages, temperatures in the system or for any general-purpose use as well.

NOTE: HADC is used by the TMU for temperature monitor, so it may be unavailable during temperature conversion. For details, refer to the [TMU and HADC](#) section.

HADC Features

The HADC supports following features:

- 12-bit ADC core with built-in sample and hold
- ENOB = 10 bit
- 4 input channels
- Throughput rates up to 1 MSPS
- Single ended operation
- External reference nominal at 1.8 V
- Analog input 0 V to 1.8 V.
- Selectable ADC clock frequency through a pre-scaler
- Conversion type adaptable to each application: allows single or continuous conversion with option of auto-scan
- Auto sequencing capability provides up to 4 *auto-conversions* in a single session. Each conversion can be programmed to select any of the available input channels.
- Four data registers (individually addressable) to store conversion values
- Number of channels: 4 for BGA and 2 for LQFP
- Auto sequencing capability provides up to 8 *auto-conversions* in a single session. Each conversion can be programmed to select any of the available input channels.

HADC Functional Description

The HADC provides the analog to digital conversion capability for general-purpose housekeeping tasks, such as voltage and temperature monitoring. The core of HADC is a 12-bit SAR ADC, providing multiple analog input channels.

The HADC has the following functionality:

Fixed and continuous conversion modes

ADC converts the input channel sequence for a fixed number of times or continuously converts an input channel sequence.

Auto scanning

All the input channels can be sampled in a sequential manner.

Channel sequence programming

The sequence of a channel can be selected by programming the channel mask register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain.

ADSP-2156x HADC Register List

The Housekeeping ADC (HADC) provides a general purpose, multi-channel successive approximation A-to-D converter. A set of registers governs HADC operations. For more information on HADC functionality, see the HADC register descriptions.

Table 30-1: ADSP-2156x HADC Register List

Name	Description
HADC_CHAN_MSK	Channel Mask Register
HADC_CTL	Control Register
HADC_DATA[nn]	Channel Data Registers
HADC_IMSK	Interrupt Mask Register
HADC_STAT	Status Register

ADSP-2156x HADC Interrupt List

Table 30-2: ADSP-2156x HADC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
122	HADC0_EVT	HADC0 Event	Edge	

ADSP-2156x HADC Trigger List

Table 30-3: ADSP-2156x HADC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
92	HADC0_EOC	HADC0 HADC0 End of Conversion	Edge

Table 30-4: ADSP-2156x HADC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

HADC Definitions

The following definitions are helpful when using the HADC module.

Auto-scan

Auto-scan is a feature which allows the multiple channels to be scanned and converted in sequence one after the other.

HADC Wakeup Time

It is the time required by the module after coming out of power down before it can start converting.

Signal-to-Noise and Distortion Ratio (SINAD)

The measured ratio of signal-to-noise and distortion at the output of the ADC. The signal is the rms amplitude of the fundamental. Noise is the sum of all non-fundamental signals up to half the sampling frequency ($f_s/2$), excluding dc. The ratio depends on the number of quantization levels in the digitization process; the more levels, the smaller the quantization noise. The theoretical signal-to-noise and distortion ratio for an ideal N-bit converter with a sine wave input is given by:

$$\text{Signal-to-(Noise + Distortion)} = (6.02 N + 1.76) \text{ dB}$$

Total Harmonic Distortion (THD)

The ratio of the rms sum to the harmonics to the fundamental.

Peak Harmonic or Spurious Noise

The ratio of the rms value of the next largest component in the ADC output spectrum (up to $f_s/2$ and excluding dc) to the rms value of the fundamental. Typically, the value of this specification is determined by the largest harmonic in the spectrum, but for ADCs where the harmonics are buried in the noise floor, it is a noise peak.

Integral Nonlinearity

The maximum deviation from a straight line passing through the endpoints of the ADC transfer function. The endpoints are zero scale, a point 1 LSB below the first code transition, and full scale, a point 1 LSB above the last code transition.

Differential Nonlinearity

The difference between the measured and the ideal 1 LSB change between any two adjacent codes in the ADC.

Offset Error

The deviation of the first code transition (00...000) to (00...001) from the ideal—that is, $GND1 + 1$ LSB.

Offset Error Match

The difference in offset error between any two channels.

Gain Error

The deviation of the last code transition (111...110) to (111...111) from the ideal (that is, $REFIN - 1$ LSB) after the offset error has been adjusted out.

Gain Error Matching

The difference in gain error between any two channels.

Power Supply Rejection Ratio (PSRR)

PSRR is defined as the ratio of the power in the ADC output at full-scale frequency, f , to the power of a 100 mV p-p sine wave applied to the ADC VDD supply of frequency, f_S . The frequency of the input varies from 5 kHz to 25 MHz. $PSRR (dB) = 10 \log(P_f/P_{f_S})$ where: P_f is the power at frequency, f , in the ADC output. P_{f_S} is the power at frequency, f_S , in the ADC output.

HADC Block Diagram

The *HADC Block Diagram* figure shows the functional blocks within the HADC and the interface to the processor core and the peripherals. The HADC supports four channels with no provision of external multiplexer.

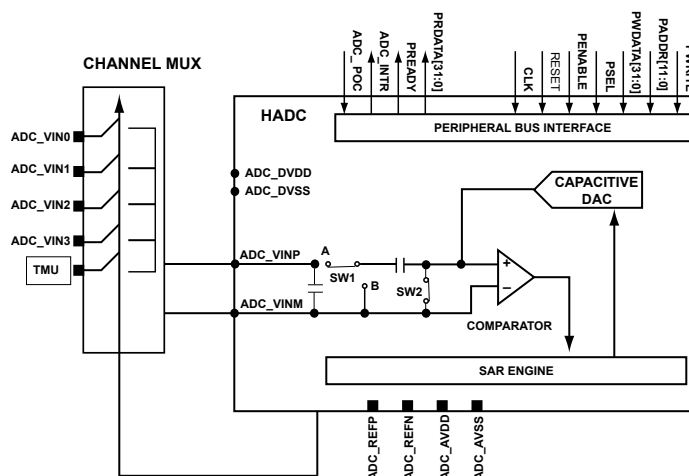


Figure 30-1: HADC Block Diagram

HADC Signal Descriptions

The *HADC Signal Descriptions* table provides descriptions of the signals used by the HADC.

Table 30-5: HADC Signal Descriptions

Signal Name	Signal Description
AVDD	ADC I/O supply
AVSS	I/O ground for analog blocks
VREFP	External reference for ADC
VREFN	Ground reference for ADC
VIN _n	Analog input at channel n

HADC Architectural Concepts

The HADC is based on a 12-bit SAR ADC that provides a simple register-based access model to obtain the results of conversion. The digital front end of the HADC provides a set of registers to configure the mode of operation, sampling frequency, and input channel selection control. The ADC supports multiple input analog channels which can be individually selected or deselected for conversion. The results of each analog channel are stored in a register. The core can access the register to read the conversion results once the conversion is complete. The HADC also provides the interrupts on completion of each channel conversion to avoid polling by the core. The following sections provide more details about the architecture of the HADC.

Converter Operation

The housekeeping ADC is a 12-bit successive approximation ADC based around a capacitive DAC. The *ADC Acquisition Phase* figure and the *ADC Conversion Phase* figure show simplified schematics of the ADC. The ADC is comprised of control logic, SAR, and a capacitive DAC. The components are used to add and subtract fixed amounts of charge from the sampling capacitor to bring the comparator back into a balanced condition. The *ADC Conversion Phase* figure shows the ADC during its acquisition phase. SW2 is closed and SW1 is in Position A. The

comparator is held in a balanced condition and the sampling capacitor acquires the signal on the selected V_{IN} channel.

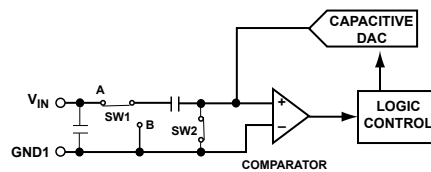


Figure 30-2: ADC Acquisition Phase

When the ADC starts a conversion (see the *ADC Conversion Phase* figure), SW2 opens and SW1 moves to Position B, causing the comparator to become unbalanced. The control logic and the capacitive DAC are used to add and subtract fixed amounts of charge to bring the comparator back into a balanced condition. When the comparator is rebalanced, the conversion is complete. The control logic generates the ADC output code.

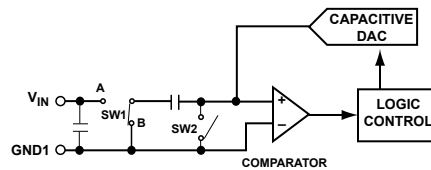


Figure 30-3: ADC Conversion Phase

Auto-Scan

The HADC features auto-scan mode where all the input channels can be sampled in a sequential manner. The number of channels enabled in auto-scan mode can be selected by programming the `HADC_CHAN_MSK` register. If the bit corresponding to the channel is set high, that particular channel is masked, and is not included in the auto-scan chain. In this way programs can sample all, none, or a selected set of channels by writing a high or a low for the individual channel. Auto sequencing allows the system to convert the same channel multiple times, allowing programs to perform oversampling algorithms.

For example, if the `HADC_CHAN_MSK` register bits [3:0] are set to 1101, then channel 0, channel 2 and channel 3 are not included in the auto-scan chain. Whether the conversion is a single or fixed number or continuous depends on the status of `HADC_CTL.CONT` bit. If this bit is low, the `HADC_CTL.FIXEDCNV` bits determine the number of sequence conversions.

The maximum number of fixed sequence conversions is four and they are enabled by default. The program must configure the `HADC_CHAN_MSK` register to enable any desired channel.

Channel Sequence Programming

The sequence of a channel can be selected by programming the `HADC_CHAN_MSK` register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain. If the program must get the conversion results from a particular channel, then the bit corresponding to that channel should be zero. The auto-scan section has more details.

ADC Transfer Function

The output coding is straight binary for the analog input channel conversion. The designed code transitions occur at successive LSB values (that is, 1 LSB, 2 LSBs, and so forth). The LSB size is $V_{REF}/4096$ for the HADC. The *ADC Transfer Function* figure shows the ideal transfer characteristic for straight binary coding.

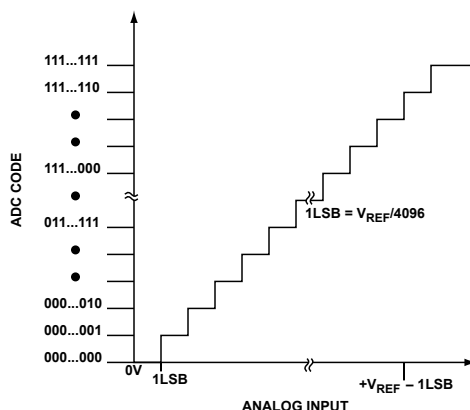


Figure 30-4: ADC Transfer Function

Results

The HADC takes 20 cycles of f_{SAMPLE} for one channel conversion. (The value of the `HADC_CTL.FDIV` bit field determines f_{SAMPLE}). The time taken to complete one sequence depends on the number of channels in the auto-scan chain. There is a latency of 1 cycle from the time the channel is selected internally to sample to the time the data is ready. After the end of each channel conversion, the data is written into the corresponding data register. An interrupt is generated (if the interrupt mask is not enabled) to signal that the data is ready for that particular channel.

HADC Operating Modes

The HADC has two modes of operation described in the following sections.

Fixed Conversion Mode

In this mode, the ADC converts the input channel sequence for a fixed number of times. The frequency is configured in the `HADC_CTL.FIXEDCNV` bit field. To use this mode, clear the `HADC_CTL.CONT` bit.

Continuous Conversion Mode

In this mode ADC continuously converts an input channel sequence, as long as `HADC_CTL.STARTCNV` bit is held high. To use this mode, set the `HADC_CTL.CONT` bit.

HADC Event Control

The HADC generates different events depending on the state of the ADC and the status of channel conversions. It can generate an event for each of the following conditions:

- When ADC is ready for conversion
- At the end of sequence conversion
- At the end of each individual channel conversion

Each of these events can generate an interrupt. To generate an interrupt on any desired event, clear the respective bit in the [HADC_IMSK](#) register.

HADC Programming Model

Following sections provide some guidelines for HADC programming.

Powering Up the HADC

To power-up the HADC, program the following bits in the [HADC_CTL](#) register.

- Deassert the [HADC_CTL.PD](#) bit (HADC power down)
- Set the [HADC_CTL.NRST](#) bit (Reset)
- Set the [HADC_CTL.ENLS](#) bit (Enable level shifters)

After deasserting [HADC_CTL.PD](#), the HADC requires a finite wake-up time (t_{WAKEUP}) before it can start converting. The HADC requires only two f_{SAMPLE} clocks from the assertion of the [HADC_CTL.NRST](#) bit before the module is ready to convert. ([HADC_CTL.PD](#) is low). Poll the [HADC_STAT.RDY](#) bit. A 1 on this bit indicates that the HADC is ready to convert data.

Enabling the HADC

Setting the [HADC_CTL.STARTCNV](#) bit enables the HADC. When this bit is kept high, the HADC can work in either continuous or fixed conversion mode. After the [HADC_CTL.STARTCNV](#) bit is set =1, the [HADC_CHAN_MSK](#) can still be re-programmed, but the new sequence only comes into effect after the current sequence conversion is complete.

ADSP-2156x HADC Register Descriptions

Housekeeping ADC (HADC) contains the following registers.

Table 30-6: ADSP-2156x HADC Register List

Name	Description
HADC_CHAN_MSK	Channel Mask Register
HADC_CTL	Control Register
HADC_DATA[nn]	Channel Data Registers
HADC_IMSK	Interrupt Mask Register

Table 30-6: ADSP-2156x HADC Register List (Continued)

Name	Description
HADC_STAT	Status Register

Channel Mask Register

The `HADC_CHAN_MSK` register provides bits that mask each channel. The LSB corresponds to channel 0, the second LSB to channel 1 and so on. If a mask bit is set, the corresponding channel is not converted.

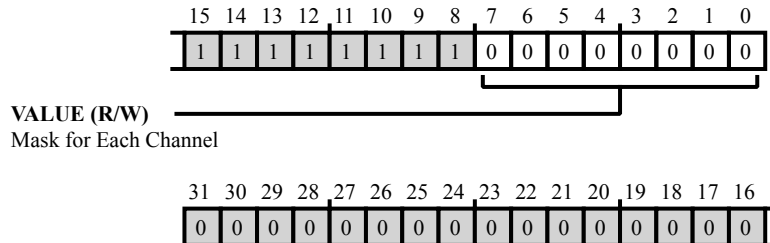


Figure 30-5: HADC_CHAN_MSK Register Diagram

Table 30-7: HADC_CHAN_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Mask for Each Channel. The <code>HADC_CHAN_MSK.VALUE</code> bit field is the mask bit for each channel. Lsb corresponds to channel 0, second LSB to channel 1 and so on. If the mask is set for a particular channel, that channel is not converted. Among the 8 channels, unused channels should be masked. The product supports only 4 or 2 channels depending on the package type i.e Program <code>CHAN_MSK[7:4]</code> to 0xF for BGA package and <code>CHAN_MSK[7:2]</code> to 0x3F for LQFP package

Control Register

The `HADC_CTL` register contains control bits that configure various module settings start or reset the module.

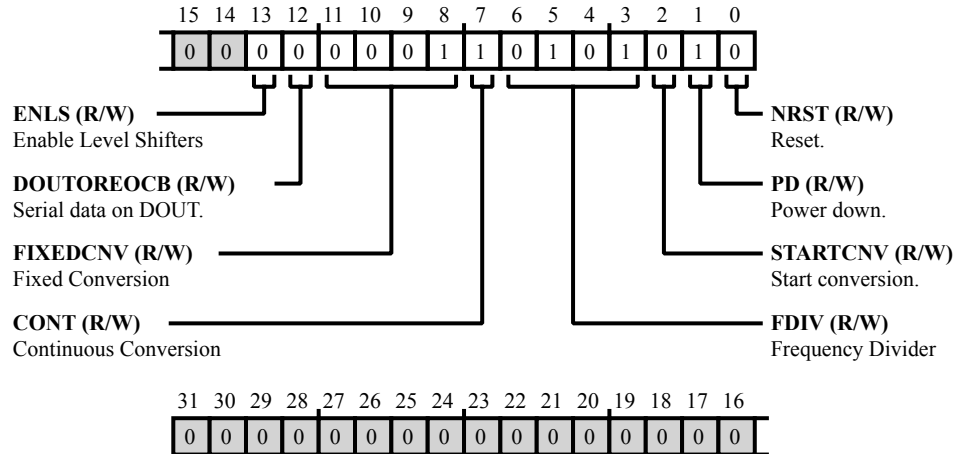


Figure 30-6: HADC_CTL Register Diagram

Table 30-8: HADC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	ENLS	Enable Level Shifters. Setting the <code>HADC_CTL.ENLS</code> bit enables the level shifters, allowing the HADC analog side (which works in the <code>VDD_EXT</code> domain) to work with the digital core and interface is (in the <code>VDD_INT</code> domain).
		0 Disable down level shifters
		1 Enable down level shifters
12 (R/W)	DOUTOREOCB	Serial data on DOUT.. If the <code>HADC_CTL.DOUTOREOCB</code> bit =1, serial data arrives on the <code>EOC_DOUT</code> pin. If this bit =0 (default) it acts as an EOC only if the external multiplexer is connected.
		0 Reserved
		1 Reserved
11:8 (R/W)	FIXEDCNV	Fixed Conversion. The <code>HADC_CTL.FIXEDCNV</code> bit configures the number of conversions = <code>FIXEDCNV</code> . This value determines how many times a sequence is converted when the HADC is in fixed conversion mode. This only applies when the <code>HADC_CTL.CONT</code> bit =0. Before changing the <code>HADC_CTL.FIXEDCNV</code> bit, clear the <code>HADC_CTL.NRST</code> bit (=0).

Table 30-8: HADC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	CONT	Continuous Conversion. When the HADC_CTL.CONT bit =0, the ADC converts a sequence for a fixed number of times. This number is configured using the HADC_CTL.FIXEDCNV bit field. When the HADC_CTL.CONT bit =1, the ADC continuously converts a given sequence, provided the HADC_CTL.STARTCNV is held high.
		0 ADC converts sequence for fixed number of times
		1 ADC continuously converts given sequence
6:3 (R/W)	FDIV	Frequency Divider. The HADC_CTL.FDIV bit field configures the $f_{SAMPLE}=f_{CLK}/(FDIV+1)$. Select f_{CLK} and HADC_CTL.FDIV values so that f_{SAMPLE} is in range of 50 kHz to 22.5 MHz. The minimum value for HADC_CTL.FDIV is 1. Before changing the HADC_CTL.FDIV bits, clear the HADC_CTL.NRST bit.
2 (R/W)	STARTCNV	Start conversion.. The HADC_CTL.STARTCNV bit needs to be set for the ADC to start converting data. If the ADC is running in non continuous mode, it is reset by hardware after the desired number of conversions is completed.
		0 No action
		1 Start converting
1 (R/W)	PD	Power down.. The HADC_CTL.PD bit powers down the analog circuitry of the ADC. After this bit returns to 0 a finite power-up time is required before the ADC can start converting data.
		0 No action
		1 Power down the analog circuitry of the ADC
0 (R/W)	NRST	Reset.. The HADC_CTL.NRST bit resets the ADC.
		0 Reset the ADC
		1 No action

Channel Data Registers

The `HADC_DATA[nn]` registers NN ranges from 0-3. Each corresponding to an ADC channel. `ADC_DATA_0` corresponds to channel 0, `ADC_DATA_1` to channel 1 and so on.

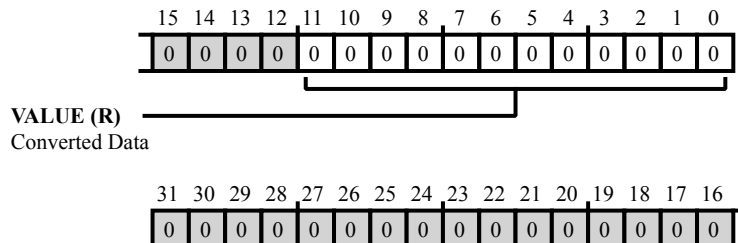


Figure 30-7: HADC_DATA[nn] Register Diagram

Table 30-9: HADC_DATA[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/NW)	VALUE	Converted Data. The <code>HADC_DATA[nn].VALUE</code> bit field contains the digital code for the sampled analog value. Each channel has its own data register.

Interrupt Mask Register

The `HADC_IMSK` register masks (disables) or unmasks (enables) the interrupts as programmed.

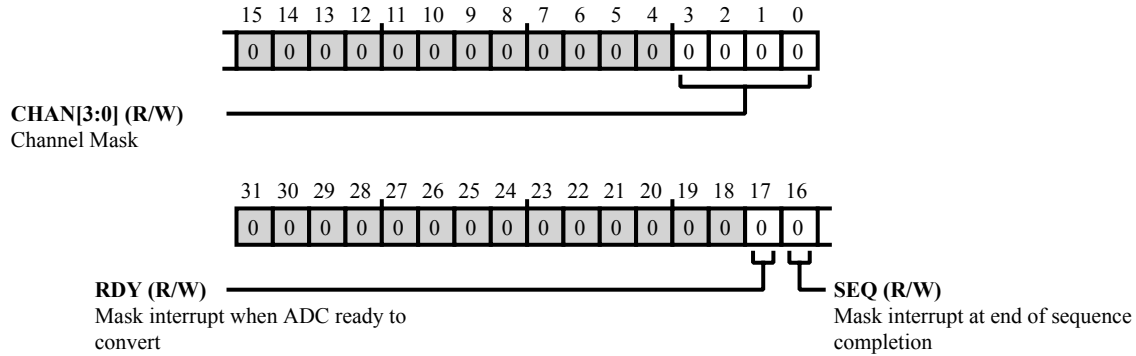


Figure 30-8: HADC_IMSK Register Diagram

Table 30-10: HADC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	RDY	Mask interrupt when ADC ready to convert. The <code>HADC_IMSK.RDY</code> bit masks the interrupt generated when ADC is ready to convert.
16 (R/W)	SEQ	Mask interrupt at end of sequence completion. The <code>HADC_IMSK.SEQ</code> bit masks the interrupt which is generated at the end of sequence completion.
		0 Interrupt is unmasked
		1 Interrupt is masked
3:0 (R/W)	CHAN	Channel Mask. The <code>HADC_IMSK.CHAN</code> bit field provides the interrupt mask bit for each channel. <i>N</i> ranges from 0-3. The MSB corresponds to channel 3, the second MSB to channel 2 and so on. If the bit is SET, interrupt is masked for the corresponding channel.

Status Register

The `HADC_STAT` register contains bits that provide status information on the HADC module.

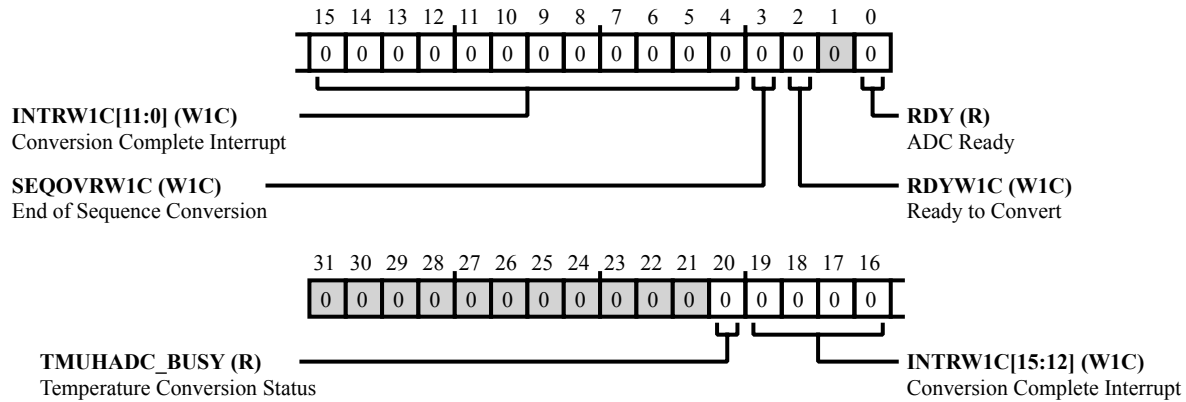


Figure 30-9: HADC_STAT Register Diagram

Table 30-11: HADC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	TMUHADC_BUSY	Temperature Conversion Status. The <code>HADC_STAT.TMUHADC_BUSY</code> bit, if high, indicates to the system that temperature conversion is ongoing. All ADC conversion requests are ignored.
19:4 (RX/W1C)	INTRW1C	Conversion Complete Interrupt. The <code>HADC_STAT.INTRW1C</code> bit field indicates when the corresponding ADC channel completes conversion. N ranges from 0-3 and the MSB corresponds to channel 3, the second MSB to channel 2 and so on. N ranges from 0-7 and the MSB corresponds to channel 7, the second MSB to channel 6 and so on. These bits are sticky and are W1C.
3 (RX/W1C)	SEQOVRW1C	End of Sequence Conversion. The <code>HADC_STAT.SEQOVRW1C</code> bit indicates the end of a sequence conversion and is a sticky status bit which is W1C.
2 (RX/W1C)	RDYW1C	Ready to Convert. The <code>HADC_STAT.RDYW1C</code> bit is the sticky version of the <code>HADC_STAT.RDY</code> bit.
0 (R/NW)	RDY	ADC Ready. The <code>HADC_STAT.RDY</code> bit is set (=1) when the ADC is ready to convert data.

31 System Security

The requirement to protect content, keys, IP and other sensitive information have become increasingly prevalent. The processor contains several modules and system elements that contribute to creating a secure operating environment for trusted code to execute.

The modules and system elements that can be used are:

- Boot Kernel
- Secure Core
- System Protection Unit (SPU)
- System Memory Protection Unit (SMPU)
- DEBUG through the Test Access Port Controller (TAPC)
- One-Time Programmable (OTP) memory
- Cryptographic Accelerators (optionally)

NOTE: This product includes security features that can be used to protect embedded non-volatile memory contents and prevent execution of non-authorized code. When security is enabled on this device (either by the ordering party or the subsequent receiving parties), Analog Devices' ability to conduct Failure Analysis on returned devices will be limited. Contact Analog Devices, Inc. for details on the Failure Analysis limitations for this device.

Security Features

The security infrastructure in the system provides the following features:

- Secure operating environment for secure code execution
- Protect sensitive IP from theft by malicious users or competitors
- Protect sensitive data (for example cipher keys)
- Allow debugging while still maintaining security

Security Functional Description

In order to provide secure operating environment in a system, it requires the involvement of multiple elements.

Boot Kernel

The Boot Kernel is the root of trust for a secure system. Since the boot kernel is developed by Analog Devices, Inc and is stored in ROM and can't be changed, it can be trusted. The boot kernel validates the authenticity of the application binary image that needs to be booted in. It also handles decrypting the binary image if it's encrypted.

Verifying the authenticity of the application binary asserts that

1. The image is not tampered with or altered
2. The image came from a trusted developer

If the image is encrypted, it ensures confidentiality since the boot image is stored on an external storage device and can be more easily read or stolen than in the part.

Once the boot kernel can verify and optionally decrypt the boot image, it can be loaded into the processor for execution. At this point, the *chain of trust* is continued with the verified application.

Secure Booting

Secure booting is when the boot kernel uses cryptographic algorithms to perform checks on the application binary and to decrypt it. When security is enabled, signature verification and optional decryption on the application binary are both performed by secure boot. See [Security Mode Configuration](#).

Secure Core

In a system with multiple master and slave resources, not everything is considered secure. There are secure and non-secure peripherals and secure and non-secure segments of memory.

For a system to be considered secure, a secure core that can access and execute instructions (verified application) from secure memory must be configured. Typically, in a single core processor, either the core is hardwired to be secure or the core can switch between secure and non-secure modes.

System Protection Unit (SPU)

The SPU in the system serves two functions. First, it acts as a gatekeeper, guarding against non-secure accesses to secure resources (peripherals). Second, it is used to define which resources in the system are secure or non-secure masters and which resources in the system are secure or non-secure slaves.

NOTE: Though the SPU can be configured by secure or non-secure master, the first steps that the verified secure application must perform are:

1. Configure the SPU as a secure slave itself so only other secure masters can configure it
2. Define and configure the secure masters and slaves using the SPU

This way, once the SPU is secure and other secure masters and slaves are configured, non-secure masters cannot tamper with the security privileges of secure masters and slaves nor can non-secure resources be changed to a secure resource.

System Memory Protection Unit (SMPU)

Similar to the SPU protecting the MMR address range for peripherals, the SMPU can guard memory ranges or pages. Memory pages can be configured as secure or non-secure. Again, like in the case of the SPU, the SMPU can guard against non-secure transaction attempts to secure memory.

NOTE: A verified application should reside in memory configured as secure. By default the SMPU has all memory configured as secure. If the program needs to update the memory protections via the SMPU, the application and its data should remain in secure memory.

Debug

The typical way of accessing a system is through the debug port via a JTAG or serial wire interface. If these access points are not secure sensitive IP such as cipher keys can be exposed and code can be changed to disable other security settings. To guard against this type of attack or security hole and still provide debugging capabilities for a developer, a debug unit with security features is used.

When the developer first receives the part, they can define a JTAG/DEBUG key that is programmed into OTP memory. Once security is enabled, the debug unit compares the key sent from the host debugger with the key inside the system. If a match occurs, debug access to secure resources are allowed. There is also a provision to bypass the loading of the JTAG/DEBUG key from the OTP memory. See Boot ROM and Booting the Processor

One-Time-Programmable (OTP) Memory

Customer programmable OTP memory is used to safely and securely store sensitive information such as cipher keys.

In a public key algorithm (for example, ECDSA which is used in secure boot), the public key is used to verify the digital signature that accompanies the application binary. The private key is used by the developer on the host development machine to create the digital signature. The public and private key pair is unique and the public key needs to be stored in non-volatile, one-time-programmable memory. If the public key is allowed to be changed then users can generate their own public/private key pairs and successfully boot in malicious code. This can either re-purpose the part or change security configurations to allow easier access to sensitive information stored elsewhere in OTP memory.

Cryptographic Accelerators

Cryptographic algorithms are mathematical tools to help provide security. Hardware engines provide some advantages but are not necessarily required. For computationally expensive operations like those used in Elliptic Curve Cryptography, like ECDSA used in secure boot, the operations can be accelerated while the core performs other tasks. Also, it's less likely that the hardware engine can be hacked to change the results.

Security Mode Configuration

Security as a feature does not necessarily be employed by the user. There are no steps to disable security, if security protection is not required. The part does not have security enabled.

To use the security features, perform the following steps:

- Generate the public/private key pair on the host development machine.¹
- Program OTP memory with the public key.²
- Program OTP memory with the decryption key if the application binary needs to be encrypted.²
- Program OTP memory with the debug/JTAG key.²
- Develop the application and sign it, creating the digital signature with the private key.¹
- If confidentiality is required, encrypt the application binary before signing it.
- Set the LOCK bit in OTP memory to enable security. After this, subsequent boots are secure boots.^{2, 3}

¹ Software tools are provided with developments tools to generate keys, sign boot streams and also perform encryption. Refer to the development tools manuals for information on usage.

²Refer to the OTP Chapter for programming the OTP and other related information.

³Refer to the Booting Chapter for more information on Secure Booting and other related information.

Status and Error Signals

In a fully functional secure system, non-secure resources should not even attempt to access a secure resource. If this does occur, then either the code has been altered or replaced with malicious code or the system contains a bug.

Errors or error events are dependant on the configuration of the SPU and SMPU, the protection units which guard against security violations. In the case of the SMPU, the error can simply be captured, captured and interrupt generated, or the access prevented without capturing any error. It is the developer's responsibility to:

1. Determine if there was an error due to a blocked access.
2. Determine how to handle a blocked access (for example fix the bug (offline), or try to use a different resource (run time)).

32 System Protection Unit (SPU)

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The SPU restricts access to certain MMRs, similar to the functionality of a semaphore.

The SPU also protects peripherals based on security settings. It is part of the overall security infrastructure of the processor.

SPU Features

The SPU has the following features:

- Write-protect system MMR from certain system masters and core masters.
- Simultaneously lock multiple peripheral configuration registers through a global lock mechanism.
- Write-protect and block access to its own write-protection registers from other system masters.
- Defined security privileges to peripherals and system resources.
- Security protection to guard secure peripheral MMRs against non-secure accesses.

SPU Functional Description

The following sections provide information on the function of the SPU.

ADSP-2156x SPU Register List

The System Protection Unit (SPU) provides a set of registers that can protect system resources from errant writes. The protection categories are global lock (protects configuration registers) and write protect register lock (protects the write protect register). For more information on SPU functionality, see the SPU register descriptions.

Table 32-1: ADSP-2156x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_SECURECHK	Secure Check Register

Table 32-1: ADSP-2156x SPU Register List (Continued)

Name	Description
<code>SPU_SECURECTL</code>	Secure Control Register
<code>SPU_SECUREC [n]</code>	Secure Core Registers
<code>SPU_SECUREP [n]</code>	Secure Peripheral Register
<code>SPU_STAT</code>	Status Register
<code>SPU_WP [n]</code>	Write Protect Register n

ADSP-2156x SPU Interrupt List

Table 32-2: ADSP-2156x SPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
186	<code>SPU0_INT</code>	SPU0 Interrupt	Level	

Peripheral Register Write Protection

The SPU has a write-protection register (`SPU_WP [n]`) associated with each peripheral. Each of these write-protection registers has the exact same bits that correspond to a particular SMMR master (for example, Core 0, MDMA). When the bits are set, the SPU locks the corresponding SMMR masters from accessing the register address space of the associated peripheral. The bits in the register can be cleared to allow access to the registers of the peripheral again. When the SPU initiates the write-protection register, any writes that are in-progress complete before the SPU blocks subsequent writes.

In the *SPU Write Protect Registers* figure, each write-protect register in the SPU is associated with a particular peripheral.

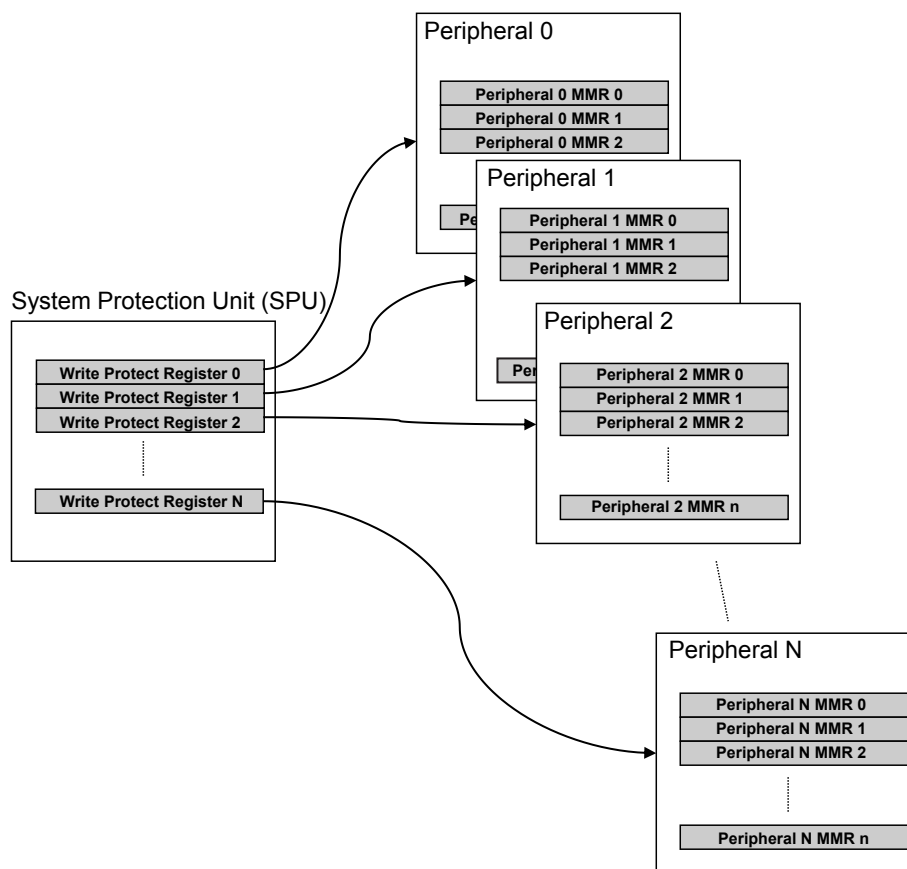


Figure 32-1: SPU Write Protect Registers

In the figure, a write-protect register in the SPU module blocks write-attempts to the MMR space of the associated peripheral. The bits in the write-protect register specify from which masters to block write-access.

NOTE: A SPU write protection register (`SPU_WP[n]`) exists for the SPU alone. If all defined bits are set in this register for the SPU, any configurations in the SPU are locked and cannot be changed. Only a system reset can restore access to the SPU.

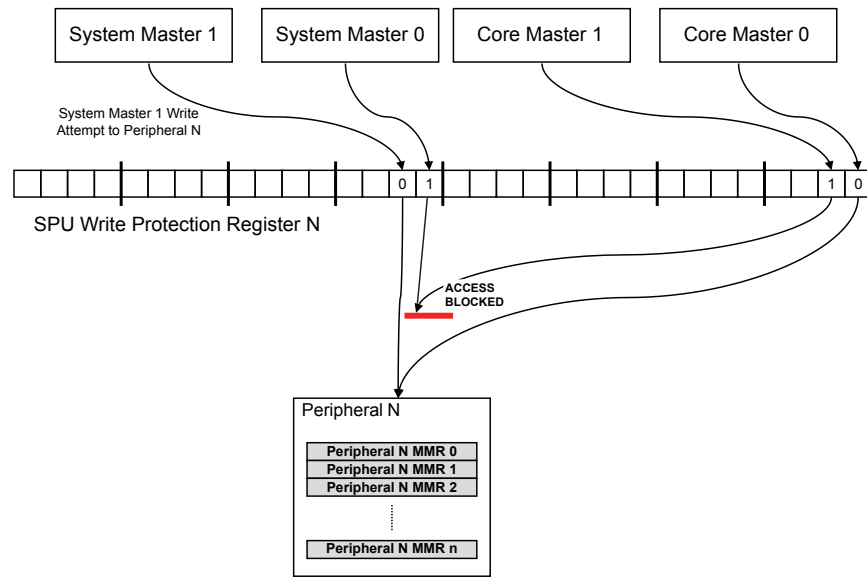


Figure 32-2: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

Global Locking

The SPU also has global locking capability. When enabled by setting `SPU_CTL.GLCK` bit field to a value other than `0xAD`, a system-wide global lock signal is active. Some peripherals have a lock enable bit in their control register. When this bit is set, the peripheral recognizes the global lock signal and blocks further write-accesses to its own control register. Access to the configuration register of the peripheral is enabled when the global lock is turned off in the SPU.

The *Global Locking* figure is a conceptual diagram. The diagram shows how the SPU module (or any peripheral) blocks any write attempts to its control register when:

- The global lock signal from the SPU is active, and
- The global lock enable bit is set in the control register of the peripheral

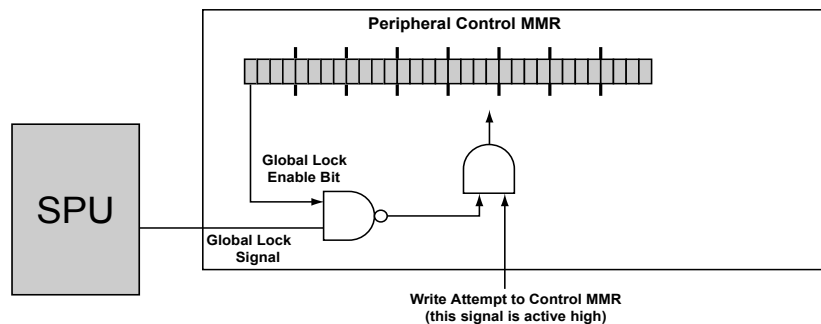


Figure 32-3: Global Locking

The SPU can write-protect its own registers. When the `SPU_CTL.WPLCK` bit is set and global locking is enabled, the SPU blocks accesses to the SPU write-protection registers. To enable write access to the write-protection registers in the SPU, disable the global locking.

SPU Block Diagram

The *SPU System-Level Block Diagram* shows a system-level block diagram of where the SPU is located in the system. It resides between the SMMR interface and the system crossbar. Depending on the configuration of the SPU write-protect registers, it can block access to some peripherals from certain SMMR masters.

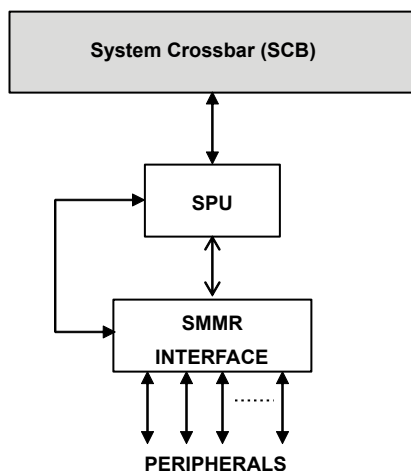


Figure 32-4: SPU System-Level Block Diagram

SPU Architectural Concepts

As shown in the block diagram, the SPU sits between the system crossbar (SCB) and the SMMR interface to the peripherals. The SPU gates any MMR access to any peripheral from any master that comes through the SCB. Depending on the configuration of the write-protection registers in the SPU, the SPU does or does not allow the MMR write to go through.

The SPU also checks whether the transaction is a secure or non-secure transaction and blocks it according to the configured security setting for the target destination. A secure master can generate secure read or secure write transactions which can access secure or non-secure slaves. A non-secure master can generate non-secure read or non-secure write transactions and can only access non-secure slaves.

SPU Event Control

The system protection unit provides write-protection against MMRs peripherals and its own write-protect registers. If a write attempt is made to any locked MMR peripheral the SPU has write-protected, it blocks the write. The SPU generates a bus error to the master that attempted the write. That master does or does not generate an event, based on the returned error.

The SPU can be configured to generate an interrupt for the write-protection violation by setting the `SPU_CTL.PINTEN` bit. The SPU can also be configured to generate an interrupt for a security violation by setting the `SPU_SECURECTL.SINTEN` bit. If either one or both bits is triggered, the `SPU_STAT.VIRQ` bit is set.

The SPU can also lock its own registers from write attempts. If a write-attempt is made to a locked register in the SPU, the SPU blocks it and records it as an error in the `SPU_STAT.LWERR` bit. Again, the SPU generates a bus error to the master that attempted the write.

The master does or does not generate an event, based on the returned error.

The SPU does not generate an event for a blocked write access to an SPU register. If the `SPU_CTL.PINTEN` bit is set, the SPU triggers an interrupt for this blocked access attempt.

The global lock is enabled by setting the `SPU_CTL.GLCK` bit to something other than `0xAD`. If the lock bit is set in that same configuration register, a peripheral can block write access to its configuration register. When the SPU blocks a write attempt, the peripheral logs and reports the failed attempt. The SPU is unaware and therefore does not provide any indication of a failed write attempt to the configuration register of the peripheral.

SPU Programming Model

The system protection unit (SPU) consists of write-protect and access-protect registers. Each one corresponds to a different peripheral instance. Bits in the write-protect registers correspond to system masters that can modify the MMR contents of the peripherals. By writing to these write-protect registers, the corresponding memory-mapped registers of the peripheral are write-protected against masters whose bits in the write-protect register are set.

The SPU globally locks the control register of the peripheral. Peripherals that support this feature have a lock enable bit in their control register. The peripheral blocks any additional write attempts to its control register from any master when:

- The global lock signal is active from the SPU, and
- The lock enable bit of the peripheral is set

If the lock enable bit of a peripheral is not set and the global lock signal is active, access to that control register of the peripheral is still allowed. To grant access again, disable the global lock signal from the SPU by writing the value `0xAD` into the `SPU_CTL.GLCK` bit field.

Another protection mechanism that the SPU offers is write-protection against the write-protection registers. If the write protect register lock bit (`SPU_CTL.WPLCK`) is set and the global lock signal is active, writes to the write-protect registers of the SPU are blocked. To reenable access to the write-protect registers in the SPU, deactivate the global lock signal by writing `0xAD` into the `SPU_CTL.GLCK` bit field.

For security, the SPU provides a set of `SPU_SECUREC[n]` registers (one for each processor core from Analog Devices) to configure their security settings. The SPU also provides a set of `SPU_SECUREP[n]` registers (one for each peripheral instance) to configure their security settings.

Enabling and Disabling the SPU

The SPU is always operating. There are no bits to enable or disable the SPU. The SPU configuration can be updated at any time. Any ongoing transactions finish before a new configuration is in effect. By default, the SPU does not write-protect any of the MMRs.

Write-Protecting the SPU

The SPU is treated like any other peripheral in the system. As such, the SPU also has an associated write-protection register. If this write-protection register is configured to block all writes from all masters, any SPU configuration remains the same until the next system reset.

Checking the Security State

In some cases while running a peripheral, an application system master does not know whether they are a secure master generating secure transactions or not. The SPU provides a means for checking the security state of the master through the `SPU_SECURECHK` register. When read by a secure master, the register reads `0xFFFFFFFF` and when read by a non-secure master, the value is `0x00000000`.

SPU Mode Configuration

The SPU can provide address range-wide protection by write-protecting the peripherals MMR address range from system MMR masters. It can also provide register wide protection using global locking. Peripherals that support this feature can enable it in their respective configuration register. When the SPU enables the global lock signal, all subsequent writes to the configuration register of the peripheral are blocked until the global lock signal is deasserted. Similarly, the write-protection registers of the SPU can be write-protected using the global lock signal as well. The SPU uses all these modes of operation together.

Locking Write-Protect Registers

Use the following steps to lock (write-protect) a register.

1. Set the `SPU_CTL.WPLCK` bit and configure the `SPU_CTL.GLCK` field to something other than `0xAD`.

The SPU write-protect registers are blocked from further write accesses.

Protecting a Peripheral

Use the following procedure to protect a peripheral.

1. Determine which peripheral needs protection and locate the corresponding write-protect register (`SPU_WP[n]`) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. Determine the SMMR masters from which the peripheral needs protection. Then, set the corresponding bit or bits in the write-protect register (`SPU_WP[n]`) for the peripheral. See the "Write-Protect and Secure Peripheral Registers" section.

After setting the write-protect register for the particular peripheral, the identified SMMR masters are blocked from writing to any MMR in the address space of the peripheral. This block remains in place until the bits in the write-protect register are cleared.

Configuring Security Privileges of a Peripheral

Use the following procedure to configure the security privileges of a peripheral.

1. Determine the peripheral and its corresponding secure peripheral register (`SPU_SECUREP[n]`) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. If the peripheral is to be a secure slave only accepting secure transactions, set bit 0 (`SPU_SECUREP[n].SSEC`).
3. If the peripheral is to be a secure master that generates secure transactions (keeping in mind not all peripherals can be masters), set bit 1 (`SPU_SECUREP[n].MSEC`).

This procedure sets the security privileges of a peripheral.

NOTE: Only a secure master can set security privileges, keeping the chain of trust intact. If a non-secure master configures the security privileges, it can undermine security protection.

ADSP-2156x SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 32-3: ADSP-2156x SPU Register List

Name	Description
<code>SPU_CTL</code>	Control Register
<code>SPU_SECURECHK</code>	Secure Check Register
<code>SPU_SECURECTL</code>	Secure Control Register
<code>SPU_SECUREC[n]</code>	Secure Core Registers
<code>SPU_SECUREP[n]</code>	Secure Peripheral Register
<code>SPU_STAT</code>	Status Register
<code>SPU_WP[n]</code>	Write Protect Register n

Control Register

The SPU control register (`SPU_CTL`) provides a global lock for configuration registers as well as control for locking the write protect (`SPU_WP[n]`) registers. It also controls the generation of an interrupt to report blocked accesses.

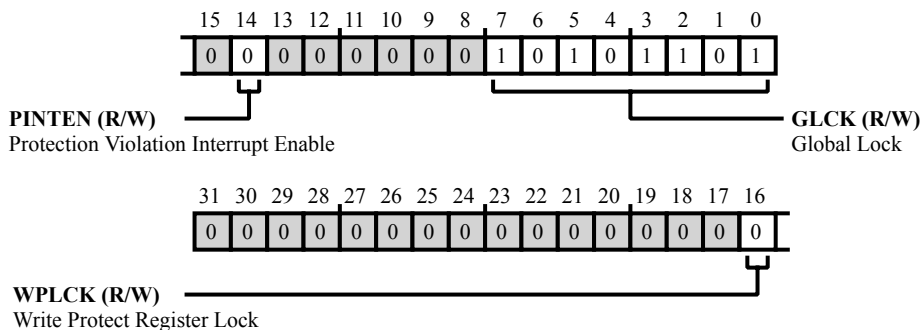


Figure 32-5: SPU_CTL Register Diagram

Table 32-4: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	WPLCK	Write Protect Register Lock. When the <code>SPU_CTL.WPLCK</code> bit is set in combination with the <code>SPU_CTL.GLCK</code> bit, writes to the SPU's write protect registers are blocked and return an error.
		0 Disable
		1 Enable
14 (R/W)	PINTEN	Protection Violation Interrupt Enable. When the <code>SPU_CTL.PINTEN</code> bit is set (=1), a block of any transaction according to the configured settings produces an interrupt.
		0 Disable
		1 Enable
7:0 (R/W)	GLCK	Global Lock. The <code>SPU_CTL.GLCK</code> controls the global lock signal. The global lock signal provides register-based write protection. Writing 0xAD to this field disables the lock, and writing any other value enables the lock.

Secure Check Register

The `SPU_SECURECHK` register reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

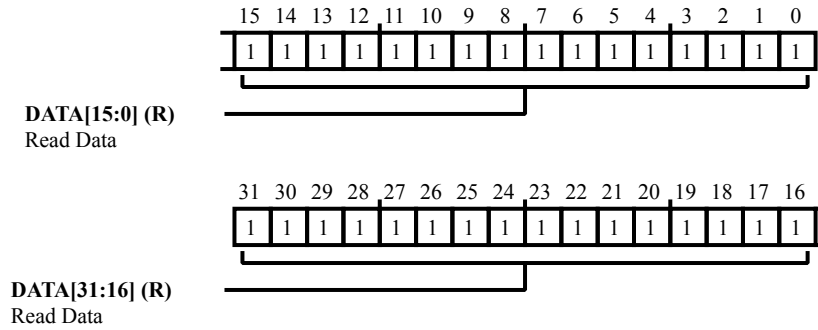


Figure 32-6: SPU_SECURECHK Register Diagram

Table 32-5: SPU_SECURECHK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Read Data. The <code>SPU_SECURECHK</code> . <code>DATA</code> bit field performs reads. Reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

Secure Control Register

The SPU Secure Control Register (`SPU_SECURECTL`) allows the user to lock write access to all the `SPU_SECUREC[n]` and `SPU_SECUREP[n]` registers as well as configure the interrupt generation in an event of a security error. It also allows bulk clear of the SSEC bits and/or MSEC bits in the `SPU_SECUREP[n]` registers.

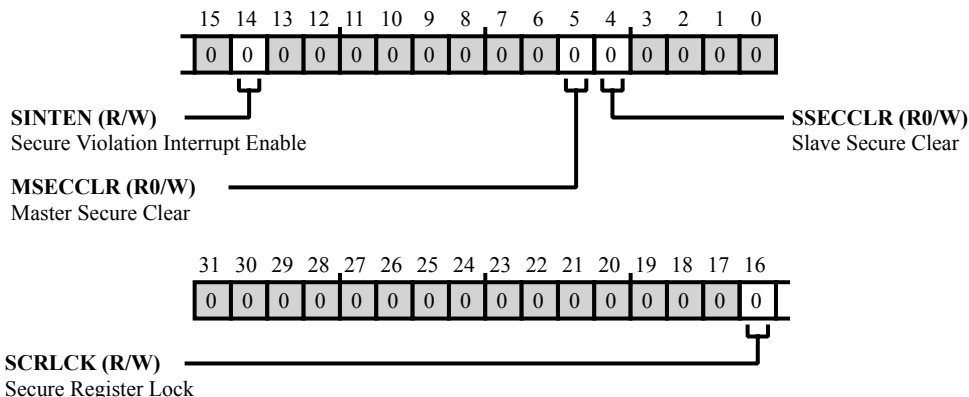


Figure 32-7: SPU_SECURECTL Register Diagram

Table 32-6: SPU_SECURECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SCRLCK	Secure Register Lock. When the <code>SPU_SECURECTL.SCRLCK</code> bit is set in combination with the <code>SPU_CTL.GLCK</code> bit, writes to the Security Configuration registers (<code>SPU_SECUREC[n]</code> and <code>SPU_SECUREP[n]</code>) are blocked and return an error which is captured in the <code>SPU_STAT.LWERR</code> bit.
		0 Disable
		1 Enable
14 (R/W)	SINTEN	Secure Violation Interrupt Enable. The <code>SPU_SECURECTL.SINTEN</code> bit generates an interrupt if a security violation was captured. Interrupt status is provided in the <code>SPU_STAT.VIRQ</code> bit.
		0 Disable
		1 Enable
5 (R0/W)	MSECCLR	Master Secure Clear. When the <code>SPU_SECURECTL.MSECCLR</code> bit is set, the <code>SPU_SECUREP[n].MSEC</code> bits in all <code>SPU_SECUREP[n]</code> registers are cleared. The <code>SPU_SECURECTL.MSECCLR</code> bit always reads back as a 0.
		0 No Action
		1 Clear All Master Secure Control Bits

Table 32-6: SPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R0/W)	SSECCLR	Slave Secure Clear. When the SPU_SECURECTL.SSECCLR bit is set, the SPU_SECUREEP[n].SSEC bits in all SPU_SECUREEP[n] registers are cleared. The SPU_SECURECTL.SSECCLR bit always reads back as a 0.
		0 No Action
		1 Clear All Slave Secure Control Bits

Secure Core Registers

A SPU register exists for every DSP core in the system. The bits enable or disable security for features in the core.

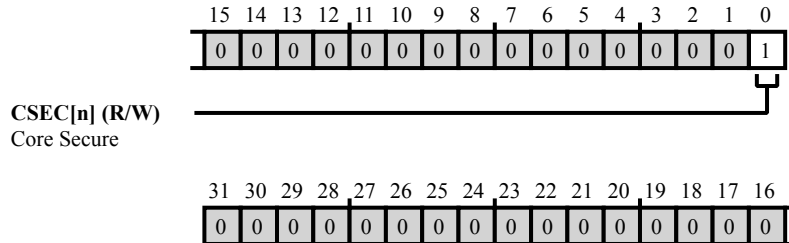


Figure 32-8: SPU_SECUREC[n] Register Diagram

Table 32-7: SPU_SECUREC[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	CSEC[n]	Core Secure. The SPU_SECUREC [n] .CSEC [n] bit controls whether non-secure accesses are allowed to L1 memory of the processor core. When =1, the core (as a slave) is set as secure meaning only secure transactions are allowed to L1.
		0 Disable
		1 Enable

Secure Peripheral Register

In the system, each `SPU_SECUREP[n]` register is assigned to a specific MMR address range associated with one peripheral. Each `SPU_SECUREP[n]` has a Slave Secure (SSEC) bit and a Master Secure (MSEC) bit. When the Slave Secure (SSEC) bit is set, the SPU will only allow Secure Masters generating secure transactions to access the peripheral's MMR address space. When the Master Secure (MSEC) bit is set, the associated peripheral will be secure and will generate secure transactions.

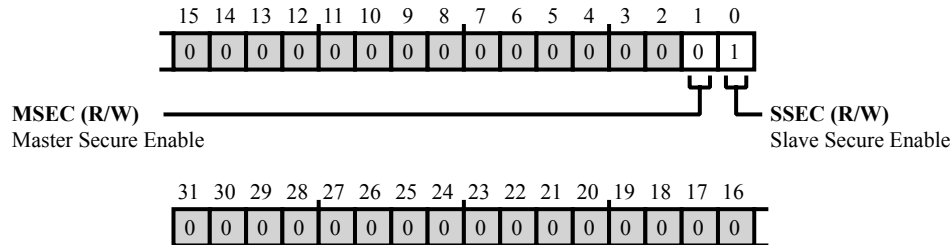


Figure 32-9: SPU_SECUREP[n] Register Diagram

Table 32-8: SPU_SECUREP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	MSEC	Master Secure Enable. The <code>SPU_SECUREP[n].MSEC</code> bit controls whether the peripheral generates secure transactions as a master. When clear (=0), the peripheral generates non-secure transactions as a master (if applicable). When set (=1), the peripheral generates secure transactions as a master.
		0 Disable
		1 Enable
0 (R/W)	SSEC	Slave Secure Enable. The <code>SPU_SECUREP[n].SSEC</code> bit controls whether the peripheral is protected from non-secure transactions. When clear (=0), the security status of the transaction is ignored. When set (=1), only secure transactions are allowed to access the address space of the peripheral and non-secure transactions are blocked.
		0 Disable
		1 Enable

Status Register

The `SPU_STAT` register indicates if there have been any errors, active interrupts and global lock status.

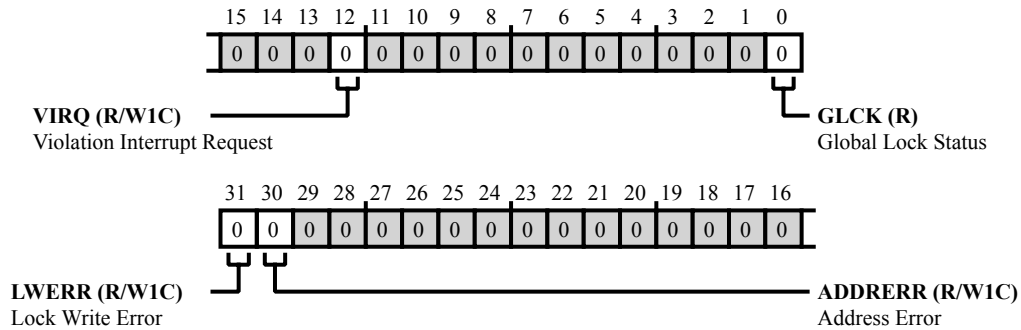


Figure 32-10: SPU_STAT Register Diagram

Table 32-9: SPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	Lock Write Error. The <code>SPU_STAT.LWERR</code> indicates whether there has been an attempted write to a register in the SPU with its lock bit (<code>SPU_CTL.WPLCK</code> or <code>SCRLCK</code>) set while <code>SPU_CTL.GLCK</code> was asserted. This bit is W1C.
		0 Inactive
		1 Active
30 (R/W1C)	ADDRERR	Address Error. The <code>SPU_STAT.ADDRERR</code> indicates whether there has been an attempted write to a read-only register or an access an invalid address in the SPU MMR address range. This bit is W1C.
		0 Inactive
		1 Active
12 (R/W1C)	VIRQ	Violation Interrupt Request. The <code>SPU_STAT.VIRQ</code> bit indicates that a security and/or protection violation has been detected and interrupt asserted. This is a W1C bit.
		0 Inactive
		1 Active
0 (R/NW)	GLCK	Global Lock Status. The <code>SPU_STAT.GLCK</code> indicates whether the global lock is enabled or disabled.
		0 Disabled (<code>global_lock=0</code>)
		1 Enabled (<code>global_lock=1</code>)

Write Protect Register n

In the system, each `SPU_WP[n]` register is assigned to a specific MMR address range associated with one peripheral. When the appropriate bits are set, writes to the peripheral from a specific master are blocked and an error is returned to the master. For more information, see the processor specific additional information for the `SPU_WP[n]` register.

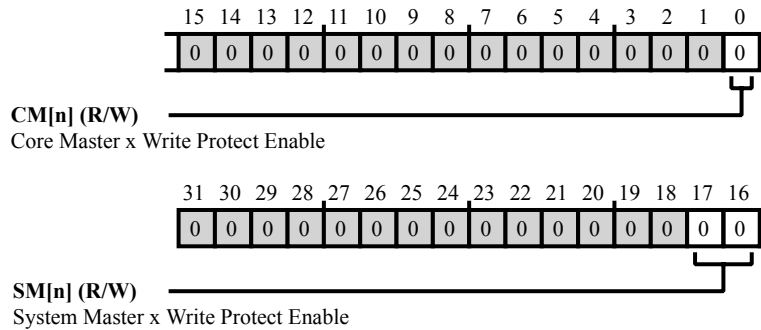


Figure 32-11: `SPU_WP[n]` Register Diagram

Table 32-10: `SPU_WP[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:16 (R/W)	SM[n]	System Master x Write Protect Enable. The <code>SPU_WP[n].SM[n]</code> bits correspond to different system masters in the system. When a particular bit is set in this field, the corresponding system master cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU.
0 (R/W)	CM[n]	Core Master x Write Protect Enable. The <code>SPU_WP[n].CM[n]</code> bits correspond to different cores in the system. When a particular bit is set in this field, the corresponding core cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU.

Write-Protect, Secure Peripheral, and Secure Core Registers

The SPU consists of a collection of write-protect registers each of which are associated with a specific peripheral or slave. The SPU also has a collection of secure peripheral registers which are also associated with specific peripherals. The table gives the write-protect register and secure peripheral number for each of the peripherals that are provided with write protection and security through the SPU. The SPU for the processor is configured with 136 write-protect registers and also 136 secure peripheral registers. The number corresponding to a peripheral correlates to both the Write-Protect register and the Secure Peripheral register.

For each processor, there are different numbers of masters that are able to access the SMMR space.

The `SPU_WP[n]` register shows which bits enable the protection against which master.

For each peripheral, there is a corresponding write-protect register, `SPU_WP[n]`, and secure peripheral register, `SPU_SECUREP[n]`. The table shows the write-protect register and secure peripheral number for each peripheral.

Table 32-11: Write-Protect Register and Secure Peripheral Number(n)

No.	Peripheral/Block Name	SPU WPn Mapping	SECUREPn Mapping	Master Capable
0	Reserved	WP0	SECUREP0	N/A
1	Reserved	WP1	SECUREP1	N/A
2	Reserved	WP2	SECUREP2	N/A
3	LP0	WP3	SECUREP3	N/A
4	LP1	WP4	SECUREP4	N/A
5	LP0 DDE	WP5	SECUREP5	Yes
6	LP1 DDE	WP6	SECUREP6	Yes
7	TWI3	WP7	SECUREP7	N/A
8	TWI4	WP8	SECUREP8	N/A
9	TWI5	WP9	SECUREP9	N/A
10	TWI0	WP10	SECUREP10	N/A
11	TWI1	WP11	SECUREP11	N/A
12	TWI2	WP12	SECUREP12	N/A
13	SPORT0A	WP13	SECUREP13	N/A
14	SPORT0B	WP14	SECUREP14	N/A
15	SPORT1A	WP15	SECUREP15	N/A
16	SPORT1B	WP16	SECUREP16	N/A
17	SPORT2A	WP17	SECUREP17	N/A
18	SPORT2B	WP18	SECUREP18	N/A
19	SPORT3A	WP19	SECUREP19	N/A
20	SPORT3B	WP20	SECUREP20	N/A
21	SPORT4A	WP21	SECUREP21	N/A
22	SPORT4B	WP22	SECUREP22	N/A
23	SPORT5A	WP23	SECUREP23	N/A
24	SPORT5B	WP24	SECUREP24	N/A
25	SPORT6A	WP25	SECUREP25	N/A
26	SPORT6B	WP26	SECUREP26	N/A
27	SPORT7A	WP27	SECUREP27	N/A
28	SPORT7B	WP28	SECUREP28	N/A

Table 32-11: Write-Protect Register and Secure Peripheral Number(n) (Continued)

No.	Peripheral/Block Name	SPU WPn Mapping	SECUREPn Mapping	Master Capable
29	UART0	WP29	SECUREP29	N/A
30	UART1	WP30	SECUREP30	N/A
31	UART2	WP31	SECUREP31	N/A
32	PORTA	WP32	SECUREP32	N/A
33	PORTB	WP33	SECUREP33	N/A
34	PORTC	WP34	SECUREP34	N/A
35	PADS	WP35	SECUREP35	N/A
36	PINT0	WP36	SECUREP36	N/A
37	PINT1	WP37	SECUREP37	N/A
38	PINT2	WP38	SECUREP38	N/A
39	Reserved	WP39	SECUREP39	N/A
40	Reserved	WP40	SECUREP40	N/A
41	WDT0	WP41	SECUREP41	N/A
42	WDT1	WP42	SECUREP42	N/A
43	CNT0	WP43	SECUREP43	N/A
44	OTP MMR	WP44	SECUREP44	N/A
45	Reserved	WP45	SECUREP45	N/A
48	TMR	WP48	SECUREP48	N/A
49	SPORT0A DDE	WP49	SECUREP49	Yes
50	SPORT0B DDE	WP50	SECUREP50	Yes
51	SPORT1A DDE	WP51	SECUREP51	Yes
52	SPORT1B DDE	WP52	SECUREP52	Yes
53	SPORT2A DDE	WP53	SECUREP53	Yes
54	SPORT2B DDE	WP54	SECUREP54	Yes
55	SPORT3A DDE	WP55	SECUREP55	Yes
56	SPORT3B DDE	WP56	SECUREP56	Yes
57	SPORT4A DDE	WP57	SECUREP57	Yes
58	SPORT4B DDE	WP58	SECUREP58	Yes
59	SPORT5A DDE	WP59	SECUREP59	Yes
60	SPORT5B DDE	WP60	SECUREP60	Yes
61	SPORT6A DDE	WP61	SECUREP61	Yes

Table 32-11: Write-Protect Register and Secure Peripheral Number(n) (Continued)

No.	Peripheral/Block Name	SPU WPn Mapping	SECUREPn Mapping	Master Capable
62	SPORT6B DDE	WP62	SECUREP62	Yes
63	SPORT7A DDE	WP63	SECUREP63	Yes
64	SPORT7B DDE	WP64	SECUREP64	Yes
65	UART0RX DDE	WP65	SECUREP65	Yes
66	UART0TX DDE	WP66	SECUREP66	Yes
67	UART1RX DDE	WP67	SECUREP67	Yes
68	UART1TX DDE	WP68	SECUREP68	Yes
69	UART2RX DDE	WP69	SECUREP69	Yes
70	UART2TX DDE	WP70	SECUREP70	Yes
71	OSPI	WP71	SECUREP71	N/A
72	Reserved	WP72	SECUREP72	N/A
73	SPI0TX DDE	WP73	SECUREP73	Yes
74	SPI0RX DDE	WP74	SECUREP74	Yes
75	SPI1TX DDE	WP75	SECUREP75	Yes
76	SPI1RX DDE	WP76	SECUREP76	Yes
77	SPI2TX DDE	WP77	SECUREP77	Yes
78	SPI2RX DDE	WP78	SECUREP78	Yes
79	SPI0	WP79	SECUREP79	N/A
80	SPI1	WP80	SECUREP80	N/A
81	SPI2	WP81	SECUREP81	N/A
82	DMC0	WP82	SECUREP82	N/A
83	DMC0 PHY	WP83	SECUREP83	N/A
84	Reserved	WP84	SECUREP84	N/A
85	L2CTL0	WP85	SECUREP85	N/A
86	SMPU L2CTL0 CL2	WP86	SECUREP86	N/A
87	SMPU L2CTL0 DL2	WP87	SECUREP87	N/A
88	SEC0	WP88	SECUREP88	N/A
89	TRU0	WP89	SECUREP89	N/A
90	SPU0	WP90	SECUREP90	N/A
91	RCU0	WP91	SECUREP91	N/A
92	CGU0	WP92	SECUREP92	N/A

Table 32-11: Write-Protect Register and Secure Peripheral Number(n) (Continued)

No.	Peripheral/Block Name	SPU WPn Mapping	SECUREPn Mapping	Master Capable
93	CGU1	WP93	SECUREP93	N/A
94	CDU0	WP94	SECUREP94	N/A
95	DPM0	WP95	SECUREP95	N/A
96	PLL0	WP96	SECUREP96	N/A
97	PLL1	WP97	SECUREP97	N/A
98	SWU_L2CTL0_CL2	WP98	SECUREP98	N/A
99	SWU_L2CTL0_DL2	WP99	SECUREP99	N/A
100	SWU SMMR	WP100	SECUREP100	N/A
101	Enhanced BW MDMA	WP101	SECUREP101	Yes
102	Max BW MDMA	WP102	SECUREP102	Yes
103	MLB0	WP103	SECUREP103	Yes
104	SWU DMC0	WP104	SECUREP104	N/A
105	SMPU DMC0	WP105	SECUREP105	N/A
106	SMPU SPI2	WP106	SECUREP106	N/A
107	MEC0	WP107	SECUREP107	N/A
108	CRC0	WP108	SECUREP108	N/A
109	CRC1	WP109	SECUREP109	N/A
110	MDMA0 DDE0 CRC0	WP110	SECUREP110	Yes
111	MDMA0 DDE1 CRC0	WP111	SECUREP111	Yes
112	MDMA1 DDE0 CRC1	WP112	SECUREP112	Yes
113	MDMA1 DDE1 CRC1	WP113	SECUREP113	Yes
114	SWU SPI2	WP114	SECUREP114	N/A
115	Reserved	WP115	SECUREP115	N/A
116	Reserved	WP116	SECUREP116	N/A
117	DAI0	WP117	SECUREP117	N/A
118	DAI1	WP118	SECUREP118	N/A
119	CRYPTO SPE	WP119	SECUREP119	Yes
120	CRYPTO PKP	WP120	SECUREP120	N/A
121	EMDMA0	WP121	SECUREP121	Yes
122	DAP ROM	WP122	SECUREP122	N/A
123	Reserved	WP123	SECUREP123	N/A

Table 32-11: Write-Protect Register and Secure Peripheral Number(n) (Continued)

No.	Peripheral/Block Name	SPU WPn Mapping	SECUREPn Mapping	Master Capable
124	Reserved	WP124	SECUREP124	N/A
125	Reserved	WP125	SECUREP125	N/A
126	Reserved	WP126	SECUREP126	N/A
127	Reserved	WP127	SECUREP127	N/A
128	ETF	WP128	SECUREP128	N/A
129	ETR	WP129	SECUREP129	N/A
130	Reserved	WP130	SECUREP130	N/A
131	Reserved	WP131	SECUREP131	N/A
132	Reserved	WP132	SECUREP132	N/A
133	TAPC MMR	WP133	SECUREP133	N/A
134	Debug Control	WP134	SECUREP134	N/A
135	SWU C0 S1	WP135	SECUREP135	N/A
136	SWU C0 S2	WP136	SECUREP136	N/A

ADSP-2156x Specific Information

Global Locking

The global lock signal from the SPU along with the peripheral lock bit can be used to provide lock functionality for the control MMR of the peripheral. The Global Lock (`SPU_CTL.GLCK`) field determines whether global lock is enabled or not. Global Lock is disabled if the `SPU_CTL.GLCK` field is 0xAD (default value), otherwise it is enabled. The following is a list of peripherals that have the global lock bit in their control MMR.

- General-Purpose IO (GPIO)
- System Event Controller (SEC0)
- Trigger Routing Unit (TRU0)
- Clock Generation Unit (CGU0)
- Clock Generation Unit1 (CGU1)
- Clock Distribution Unit (CDU0)
- Dynamic Power Management (DPM)
- Reset Control Unit (RCU0)
- System Protection Unit (SPU0)
- L2 Memory Controller (L2CTL0)

33 Security Packet Engine (PKTE)

The PKTE is a security packet engine designed to off-load the host processor to improve the speed of applications requiring cryptographic processing. The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation.

PKTE Features

The PKTE has the following features.

- Hardware assisted processing for the cryptographic ciphers, hashes, and pseudo-random number generation
- Header and trailer processing for Internet security protocols
- DMA capability to move data in and out of the engine efficiently and to allow the engine to run autonomously while moving data
- Interrupt controller to signal module status and errors
- Clock manager for enabling or disabling different features to save power

NOTE: Not all algorithms, decrypt, and hash functions and extra features are available on all product models. For complete information on included features, see the product-specific data sheet.

NOTE: This packet engine provides support for various network security protocols by processing headers and trailers as well as accelerating cryptographic functions. Not all processors have direct support for Ethernet. As such, the packet engine can still be used if Ethernet is indirectly used.

PKTE Functional Description

The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation. The following sections describe these functional blocks.

ADSP-2156x PKTE Register List

The Security Packet Engine (PKTE) provides security-related features. A set of registers governs PKTE operations. For more information on PKTE functionality, see the PKTE register descriptions.

Table 33-1: ADSP-2156x PKTE Register List

Name	Description
PKTE_ARC4STATE_ADDR	Packet Engine ARC4 State Record Address
PKTE_ARC4STATE_BUF	Starting Entry of 256-byte ARC4 State Buffer
PKTE_BUF_PTR	Packet Engine Buffer Pointer Register
PKTE_BUF_THRESH	Packet Engine Buffer Threshold Register
PKTE_CDRBASE_ADDR	Packet Engine Command Descriptor Ring Base Address
PKTE_CDSC_CNT	Packet Engine Command Descriptor Count Register
PKTE_CDSC_INCR	Packet Engine Command Descriptor Count Increment Register
PKTE_CFG	Packet Engine Configuration Register
PKTE_CLK_CTL	PE Clock Control Register
PKTE_CONT	PKTE Continue Register
PKTE_CTL_STAT	Packet Engine Control Register
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers

Table 33-1: ADSP-2156x PKTE Register List (Continued)

Name	Description
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_ARC4IJPTR	ARC4 i and j Pointer Register
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

ADSP-2156x PKTE Interrupt List

Table 33-2: ADSP-2156x PKTE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
111	PKTE0_IRQ	PKTE0 Interrupt	Level	

PKTE Definitions

Command Descriptor

An 8-word structure that is either written directly into the packet command MMR set or is placed in a Command Descriptor Ring (CDR) in the processor memory. The packet engine sequentially processes the structure. The command descriptor contains the information that varies for every packet. This information includes pointers to the SA record, the state information, the source packet, and the destination packet.

Command Descriptor Ring (CDR)

A circular contiguous portion of memory which is used to manage one or more command descriptions for the packet engine.

Result Descriptor

When the packet engine completes the processing of a packet, it writes a result descriptor with the state information. The result descriptor can be read directly from the result register set or from the Result Descriptor Ring (RDR) in the processor memory.

Result Descriptor Ring (RDR)

A circular contiguous portion of memory which holds the mirror or copy of the CDR but contains the result descriptors. The RDR and CDR can be overlaid on top of each other.

Security Association (SA) Record

A structure that contains the remainder of the information the packet engine requires to process a packet. Most of the information fields in the SA record such as the key and encryption mode are static for the lifetime of the association. The fields do not require frequent manipulation by the processor core. The SA record non-static fields are the sequence number and sequence number mask. The SA record can have a corresponding state record for saving results from the current operations that are useful for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

Cipher

A method or algorithm to encrypt or decrypt information

Hash

A cryptographic hash is a function that takes an arbitrary block of data and returns a fixed-size bit string. Four main properties define the function:

- It is easy to compute a hash value for any given input

- It is infeasible to generate the original input from a given hash
- It is infeasible to modify the input without changing the resulting hash
- It is infeasible to find two different inputs that result in the same hash

Autonomous Ring Mode (ARM)

Mode of operation in which most of the parameters as well as the data are set up in memory and moved to the engine for configuration and processing through DMA.

Target Command Mode (TCM)

Mode of operation where some parameters are set up in memory and moved into the packet engine through DMA while the other parameters are directly written to the registers. DMA moves the input and output data in and out of the engine.

Direct Host Mode (DHM)

Mode of operation that does not use DMA. All parameters are directly written to and read from the MMRs. The input and output are written to and read from the FIFO buffers.

Cipher Module

The cipher module does the symmetric encrypt or decrypt operations for:

- Data Encryption Standard (DES)
- Triple-DES
- ARC4
- Advanced Encryption Standard (AES) algorithms

The cipher module supports standard modes for DES and AES that include Electronic Code Book (ECB) and Cipher Block Chaining (CBC). The key size for DES is 56 bits, for Triple-DES is 168 bits. The AES module also provides support for AES counter-modes for IPsec and SRTP. All AES modes can use key sizes of 128 bits and 192/256 bits. Key scheduling is automatic and done in parallel with the encrypt or decrypt operation.

Hash Module

The hash module is tightly coupled with the encrypt or decrypt module and provides hardware accelerated one-way hash functions. Operations that combine both hash and encrypt or decrypt functions are provided to reduce processing time for data that needs both applied. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of hashing and the start of encryption to support protocols such as IPsec or SRTP. The HMAC keyed hashing mechanism is supported

for MD5, SHA-1, SHA-2-224 and SHA-2-256. The SSL-MAC is supported for MD5 and SHA-1. A second AES-CBC module for the hash function enables parallel processing for AES-CCM, a combined hash and encrypt algorithm.

Pseudo-Random Number Generator

Cipher algorithms that operate in CBC mode or counter-mode, require an IV. This IV must not be secret; however the IV must be unpredictable and unique for each execution of the encryption process. Pseudo-random number generators are deterministic algorithms that output statistically independent and unbiased numbers. True random number generators are non-deterministic and use the randomness that occurs in a physical process. The packet engine incorporates an ANSI X9.31 compliant Pseudo Random Number Generator (PRNG) that it can use to generate unique IVs using strong encryption. The ANSI X9.31 PRNG is defined as part of the ANSI X9 standards that are used to secure financial transactions. The function can also be used for pseudo-random number generation as part of an implementation of the digital signature standard described in NIST FIPS PUB 186-2.

The PRNG function, as defined by ANSI X9.31, is based on the AES cipher. This section describes the function to promote understanding of the different inputs and outputs of the PRNG function itself.

NOTE: The PRNG in the packet engine is only based on the AES cipher with 128-bit keys. Other ciphers and key lengths are not supported for the PRNG based on ANSI X9.31.

Let $e \times K(Y)$ represent the AES encryption of Y under the key K.

The PRNG function uses three inputs:

- K, a 128-bit key
- V, a 128-bit seed value
- DT, a 128-bit date/ time vector which is updated on each iteration

The intermediate value I is the result of an AES encryption of the data and time vector under key K.

$$I = e \times K(DT)$$

That value I is then XOR-ed with the seed V and AES encrypted under key K. The result R is the output of the PRNG function.

$$R = e \times K(I \text{ XOR } V)$$

A new seed value V is generated from the AES encryption of the result R XOR'ed with the intermediate value I under the key K.

$$V = e \times K(R \text{ XOR } I)$$

The PRNG function is deeply integrated inside the datapath of the packet engine. The function is controlled indirectly through the PRNG mode bits in the `PKTE_CTL_STAT.PRNGMD` bit field of the command descriptor and the IV source selection bits in the `PKTE_SA_CMD0.IVSRC` bit field of the SA record.

The PKTE module supports four different modes.

1. Load IV from PRNG for the current operation: `PKTE_CTL_STAT.PRNGMD = 0b00` and `PKTE_SA_CMD0.IVSRC = 0b11`.
2. PRNG init mode initializes the PRNG with a key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD = 0b01`.

Before the PRNG function can be used, it must be initialized with a key, seed, and date/time value. At initialization, the key, seed, and date/time values are programmed. Other PRNG operations do not change the key, however the seed, and date/time values are updated (not re-programmed). The date/time is updated every 128 system clock cycles.

The date/time is a 128-bit value with randomly distributed number of ones and zeros. It must not be all zeros.

The *SA Record for PRNG Init Operation* table shows how the key, seed and date/time values are loaded into the PKTE registers for initialization.

Table 33-3: SA Record for PRNG Init Operation

Parameter	SA Field	Description
K	SA_KEY0	PRNG key [127:96]
	SA_KEY1	PRNG key [95:64]
	SA_KEY2	PRNG key [63:32]
	SA_KEY3	PRNG key [31:0]
V	SA_IDIGEST0	PRNG seed [127:96]
	SA_IDIGEST1	PRNG seed [95:64]
	SA_IDIGEST2	PRNG seed [63:32]
	SA_IDIGEST3	PRNG seed [31:0]
DT	SA_ODIGEST0	PRNG date/time [127:96]
	SA_ODIGEST1	PRNG date/time [95:64]
	SA_ODIGEST2	PRNG date/time [63:32]
	SA_ODIGEST3	PRNG date/time [31:0]

3. PRNG generate mode generates pseudo-random data on initialized key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD = 0b10`.

The PRNG function can be used to generated pseudo-random data for other purposes than IVs. For this mode, the PRNG must have been initialized once with the PRNG init mode.

The PRNG generate mode uses the initialized key and unique changing date/time value as inputs. The pseudo-random data is output to the output buffer of the packet engine.

The `LEN` field in the command descriptor indicates the amount of pseudo random data that is generated in multiples of 16 bytes. The maximum is $255 \times 16 = 4080$ bytes.

In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

Directly after the PRNG generate mode, a new pseudo-random number is generated and available for the next operation that uses the option `PKTE_SA_CMD0.IVSRC = PRNG`.

4. PRNG test mode generates pseudo-random data on initialized key, seed, and input (test) data:

`PKTE_CTL_STAT.PRNGMD = 0b11`.

The PRNG test mode can be used to test the correctness of the PRNG function. This mode is similar to the PRNG generate mode, except that the data is read from the input buffer of the packet engine, instead of the date/time value.

For this mode, the PRNG must have been initialized once with the PRNG Init mode.

The `LEN` field in the command descriptor indicates the amount of pseudo-random data to be generated in multiples of 16 bytes. The maximum is limited to the `LEN` field in bytes.

In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In the direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

Directly after the PRNG test mode, a new pseudo-random number is generated and available for the next operation that uses the option `PKTE_SA_CMD0.IVSRC = PRNG`.

Packet Engine Processing Details

This section describes data processing through the packet engine. It describes padding and supported algorithms for each protocol.

A valid Security Association (SA) must be created before packet processing can start. A formatted SA record must reside in memory and be accessible to the packet engine. The host processor application is responsible for these tasks.

Crypto Padding

Padding is the process of adding data to fill-out a fixed-size plain text data structure. Three factors determine when to use a pad field:

1. If a block cipher encryption algorithm is used, a pad field is used to expand the plain text to a multiple of the block size.
2. Padding can be used to ensure that the cipher text terminates on an n-byte boundary.
3. Padding can conceal the actual length of the payload.

To facilitate peak encrypt or decrypt performance, the packet engine supports the following most commonly used padding functions in hardware:

1. Pad generation and insertion of pad bytes to the end of plain text prior to encryption, for outbound operations.

2. Pad verification to check for correct padding after decrypting a packet for inbound operations.
3. Pad consumption to strip the pad bytes from the plain text data after decrypting a packet, for inbound operations.

Pad Generation and Insertion

The pad type and quantity of bytes the packet engine inserts depends on the plain text length and the value of the following fields:

- `PKTE_SA_CMD0.PADTYPE` and `PKTE_SA_CMD0.ETXPAD` defines the type of padding
- `PKTE_CTL_STAT.PADVAL` defines a value that is inserted in the pad
- `PKTE_SA_CMD0.CIPHER` enforces a certain pad alignment
- `PKTE_SA_CMD1.CIPHERMD` enforces a certain pad alignment
- `PKTE_SA_CMD0.SCPAD` allows stream ciphers to be padded
- `PKTE_CTL_STAT.PADCTLSTAT` controls the pad alignment

The `PKTE_CTL_STAT.PADCTLSTAT` bit field of the result descriptor returns the total number of inserted pad bytes.

Pad Types

The pad type bit field (`PKTE_SA_CMD0.PADTYPE`) and the extended pad bit (`PKTE_SA_CMD0.ETXPAD`) select the pad type for the extended protocol group. The packet engine can generate different pad types in hardware as described in the *Pad Examples* table.

The `PKTE_CTL_STAT.PADVAL` bit field, together with the number of pad bytes, defines the value that is inserted in the pad. The format of the pad and the use of this field is best explained in an example (see the *Pad Examples* table).

For the IPsec pad type, this field holds the value that is inserted into the next header field (in the ESP trailer) of the innermost operation's header. For the Constant pad type or the Constant SSL pad type, this field holds the inserted fixed constant pad value. For all other pad types, this field is not used and must be zero.

Table 33-4: Pad Types

Pad Type	Value	Description
IPSec	0b000	Append 0 to 255 pad bytes, followed by a pad length byte and a next header byte. The first pad byte appended to the plain text is numbered 1, with subsequent pad bytes making up a monotonically increasing sequence: 1, 2, 3 and up. Append the pad length field that indicates the number of pad bytes (0–255), where a value of zero indicates no pad bytes present. Append the next header byte as specified in the <code>PKTE_CTL_STAT.PADVAL</code> field of the command descriptor. A minimum of 2 bytes are appended; zero pad bytes plus the pad length byte plus the next header byte, in which case the <code>PKTE_CTL_STAT.PADCTLSTAT</code> field in the result descriptor returns 0x02. A maximum of 257 bytes can be appended, in which case the <code>PKTE_CTL_STAT.PADCTLSTAT</code> field in the descriptor returns 0x01.
PKCS#7	0b001	Appends 1–128 pad bytes with a pad byte value equal to the pad length (1–128).
Constant	0b010	Appends 0–255 pad bytes of a user-specified character to the plain text data. This character is specified in the <code>PKTE_CTL_STAT.PADVAL</code> field of the command descriptor.
Zero	0b011	Appends 0–255 pad bytes of 0x00 to the plain text data.
Constant SSL	0b110	Appends 0–255 pad bytes of a user-specified character to the plain text data, followed by a ‘pad length’ byte (0–255). This character is specified in the <code>PKTE_CTL_STAT.PADVAL</code> field of the command descriptor. A total of 256 bytes can be appended, in which case the pad field returns 0x00.

For example, the *Pad Examples* table shows the appended pad for any of the pad types for an outbound (encrypt) operation. The table shows a plain text input of 2 bytes using the 8-byte block cipher crypto-algorithm DES-ECB and a `PKTE_CTL_STAT.PADVAL` field value of 0xAA.

Table 33-5: Pad Examples

Pad Type	Pad field (extended to Plain Text)	<code>PKTE_CTL_STAT</code>
IPSec	0x01, 0x02, 0x03, 0x04, 0x04, 0xAA	0x06
PKCS#7	0x06, 0x06, 0x06, 0x06, 0x06, 0x06	0x06
Constant	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA	0x06
Zero	0x00, 0x00, 0x00, 0x00, 0x00, 0x00	0x06
Constant SSL	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0x05	0x06

Pad Length

The *Pad Alignment* table lists the alignment (boundaries) to which the packet engine will pad, based on:

- The selected crypto-algorithm
- Crypto mode
- The value of the pad stream cipher bit
- The value of pad control

The minimum number of inserted pad bytes depends on the cipher algorithm, selected using the `PKTE_SA_CMD0.CIPHER` bit field, and the cipher mode, selected using the `PKTE_SA_CMD1.CIPHERMD` bit field.

For block ciphers, the plain text data is always (as a minimum) padded to the next block boundary. More pad bytes beyond the algorithm or protocol alignment requirements can be inserted using the pad control (`PKTE_CTL_STAT.PADCTLSTAT`) in the command descriptor. This feature can be used for *traffic flow security* to conceal the number of plain text bytes in an encrypted packet.

Encrypt operations that use block ciphers have minimum pad requirements based on their block size. The packet engine enforces a minimum pad alignment for block ciphers according to the *Pad Alignment* table. For ESP out-bound operations, the minimum pad alignment is forced to 4 bytes.

For stream ciphers and null crypto the data is never padded when the stream cipher pad bit (`PKTE_SA_CMD0.SCPAD`) = 0. When `PKTE_SA_CMD0.SCPAD` = 1 the plain text data is padded to the length as defined by the `PKTE_CTL_STAT.PADCTLSTAT` bits in the command descriptor.

NOTE: The SSL protocol does not allow padding to exceed the ciphers block length. This length is 8 bytes for DES/Triple-DES and 16 bytes for AES. For SSL, the packet engine does not enforce this pad alignment value. The host processor must ensure that the `PKTE_CTL_STAT.PADCTLSTAT` bit field is configured correctly.

Table 33-6: Pad Alignment

Pad Control PADCTLSTAT = PKTE_CTL_STAT[31:24]			0x00	0x01* ¹	0x02	0x04	0x08	0x10	0x20	0x40	0x80* ²
Crypto Algorithm	Crypto Mode	Pad Stream Ciphers	%8	%1	%4	%8	%16	%32	%64	%128	%256
DES	ECB, CBC	N/A	8	8	8	8	16	32	64	128	256
AES	ECB, CBC	N/A	16	16	16	16	16	32	64	128	256
	CTR, ICM with ESP	N/A	8	4	4	8	16	32	64	128	256
	CTR, ICM no ESP	no	no	0	0	0	0	0	0	0	0
yes		yes	8	0	4	8	16	32	64	128	256
NULL	with ESP	N/A	8	4	4	8	16	32	64	128	256
	no ESP	no	8	0	0	0	0	0	0	0	0
		yes	yes	8	0	4	8	16	32	64	128
ARC4	with ESP	N/A	8	4	4	8	16	32	64	128	256
	no ESP	no	0	0	0	0	0	0	0	0	0
		yes	yes	8	0	4	8	16	32	64	128

*1 If `PKTE_CTL_STAT.PADCTLSTAT` is configured for no padding (0x01), it does not mean that no padding bytes are inserted. When PKCS#7 padding is selected, a pad length field with a value =1 is inserted. When SSL or TLS padding is selected, a pad length field with a value =0 is inserted. When IPsec padding is selected, a pad length field is forced to (`PKTE_CTL_STAT.PADCTLSTAT=0x20`). When zero pad and constant pad are selected, no pad bytes are inserted.

*2 Pad type PKCS#7 supports a maximum length of 128 pad bytes, so the packet engine overrules a 256-byte alignment (`PKTE_CTL_STAT.PADCTLSTAT=0x80`) to a 128-byte boundary (`PKTE_CTL_STAT.PADCTLSTAT=0x40`).

The packet engine does not constrain the pad type that is used for an operation; any pad type can be used for each operation. The user must be aware that some protocol specifications only allow specific pad types. The SRTP specification does not have padding defined as padding performed by RTP. The host must pad the RTP packet.

The host software can implement padding that is not supported in hardware. In this case, the host must select the *zero pad* type and set the `PKTE_CTL_STAT.PADCTLSTAT` bit field in the packet descriptor to zero (no padding). The hardware padding engine does not add any bytes. When using a block cipher, the host must insert pad bytes. Then, the data to be encrypted (plain text and pad bytes) are a multiple of the block ciphers boundary. For stream ciphers, any number of pad bytes can be added.

Pad Verification and Consumption

The packet engine can validate a pad type against the expected values. The value of the following bits controls the pad verify function:

- `PKTE_SA_CMD0.PADTYPE` and `PKTE_SA_CMD0.EXTPAD` define the type of padding
- `PKTE_SA_CMD0.CIPHER` enforces a certain pad alignment
- `PKTE_SA_CMD1.CIPHERMD` enforces a certain pad alignment
- `PKTE_SA_CMD0.SCPAD` allows stream ciphers to be padded

When packet processing is complete, the status byte in the first word of the result descriptor reports the pad verification status. Refer to the [Table 33-28 Extended Error Codes - Status Encoding](#) table.

The `PKTE_CTL_STAT.PADCTLSTAT` bits in the first word of the result descriptor return the total number of detected pad bytes and returns zero for a pad verify error.

When the IPsec pad type is selected, the `PKTE_CTL_STAT.PADVAL` bit field of the result descriptor returns the next header field. For IPsec ESP outbound operations, this field returns the decimal value 50. For IPsec inbound operations and basic inbound operations that use the IPsec pad mode, the packet engine returns the next header field it detects on the header of the innermost operation. This value is typical for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other inbound operations, the returned pad value is zero.

Pad verification is performed for inbound (decrypt) operations that use IPsec, TLS/DTLS or PKCS#7 pad type:

- In combination with a block cipher algorithm: DES-ECB, DES-CBC, AES-ECB, AES-CBC,
- In combination with a stream cipher and stream cipher padding `PKTE_SA_CMD0.SCPAD` enabled: AES-CTR, AES-ICM and ARC4
- In combination with null-crypto (no encryption).

Pad Types

The `PKTE_SA_CMD0.PADTYPE` bit field and the extended pad `PKTE_SA_CMD0.EXTPAD` bit selects the pad type for the extended protocol group pad type. The packet engine can verify the different pad types in hardware as described in the *Pad Types* table.

The constant and zero pad types are not verified since they do not include a pad length field. The SSL pad type is not verified since it does not have a defined pattern.

Table 33-7: Pad Types

Pad Type	SA_CMD0[18,7:6]	Description
IPSec	0b000	<p>Verify that the pad field includes 0–255 pad bytes, followed by a correct pad length and a next header byte.</p> <p>Verify that pad bytes appended to the plain text are an incremental count, starting at one.</p> <p>Verify that the pad length field is the number of pad bytes (0–255), where a value of zero indicates no pad bytes present.</p> <p>Verify that a next header byte is present as the last byte of the packet, the value is not verified. This is after removal of the ICV. The total number of detected pad bytes is returned in the <code>PKTE_CTL_STAT.PADCTLSTAT</code> field in the result descriptor.</p> <p>NOTE: A minimum of 2 bytes must be present, zero pad bytes plus the pad length byte plus the next header byte. In this case the <code>PKTE_CTL_STAT.PADCTLSTAT</code> field in the descriptor returns 0x02. A maximum of 257 bytes can be present, in which case the <code>PKTE_CTL_STAT.PADCTLSTAT</code> field in the result descriptor returns 0x01. The value of the next header byte is returned in the <code>PKTE_CTL_STAT.PADVAL</code> field in the result descriptor.</p>
PKCS#7	0b001	<p>Verify that the pad field includes 1–128 pad bytes, with a pad byte value equal to the pad length (1–128).</p>

When a block cipher is used and the payload is not padded to the nearest block size boundary, as required by the protocol, a *block size error* is generated for all pad types.

Pad Consumption

The packet engine can optionally consume the decrypted pad bytes for an inbound operation that uses the IPsec, SSL, TLS/DTLS, or PKCS#7 pad types. The pad types constant and zero are not consumed since they do not include a pad length field.

Pad consumption (or stripping) is selected on a flow-by-flow basis in the SA-record with the `PKTE_SA_CMD1.CPYPAD` bit. When this bit is set, the length returned in the `LEN` field of the result descriptor is the total length of the plain text including the pad. When the `PKTE_SA_CMD1.CPYPAD` is disabled, the detected pad length, as returned in the `PKTE_CTL_STAT.PADCTLSTAT` bits, is subtracted from the total length and then returned in the `LEN` field of the result descriptor.

The pad is always written to the result packet buffer in memory. When the `PKTE_SA_CMD1.CPYPAD` bit is disabled, only the result length is corrected.

Crypto and Hash Algorithms

The packet engine supports a wide range of crypto and hash algorithms to accelerate basic operations and protocol operations. These algorithms are:

- Basic Encrypt and Basic Decrypt Operations
- Basic Hash Operations
- Basic Encrypt-Hash and Basic Hash-Decrypt Operations
- IPSec ESP Operations
- SRTP Operations

The following tables provide allowed algorithm combinations. Those algorithms not listed in the tables are invalid and can give unexpected results.

NOTE: Not all crypto and hash algorithms are available on all product models. For information on algorithm availability, see the product-specific data sheet.

Table 33-8: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
DES, Triple-DES	ECB, CBC
AES	ECB, CBC, CRT, ICM
NULL	—

Table 33-9: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
SHA-1	Basic hash, HMAC, SSL-MAC
SHA-224	Basic hash, HMAC
SHA-256	Basic hash, HMAC
NULL	—

Table 33-10: Algorithms for Basic Encrypt-Hash and Basic Hash-Decrypt Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	ECB, CBC	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—

Table 33-10: Algorithms for Basic Encrypt-Hash and Basic Hash-Decrypt Operations (Continued)

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	ECB, CBC, CRT, ICM	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—
NULL	—	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC

Table 33-11: Algorithms for IPsec ESP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	—

Table 33-12: Algorithms for Basic SSL and Extended SSL Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—

Table 33-12: Algorithms for Basic SSL and Extended SSL Operations (Continued)

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
ARC4	Stateful	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
NULL	—	SHA-1	SSL-MAC
	—	MD5	SSL-MAC
	—	NULL	—

Table 33-13: Algorithms for Basic TLS, Extended TLS and DTLS Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
ARC4 ¹	Stateful	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	—	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—

1 Only for Basic TLS and Extended TLS

Table 33-14: Algorithms for SRTP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	ICM	SHA-1	HMAC
NULL	—	SHA-1	HMAC

IV Processing

An initialization vector (IV) is necessary to start a cipher stream or a block cipher in any of the streaming modes of operation. The IV must be unique for each packet. The IV ensures that all cipher texts are unique even if produced by the same encryption key. This functionality prevents every packet from needing a unique encryption key.

Depending on the packet engine operation, the IV can be loaded from different sources. The IV format depends on the algorithm and the source of the IV. The *Format of the IV* table provides an overview of all IV formats.

Table 33-15: Format of the IV

Algorithm	IV Source (<code>PKTE_SA_CMD0.IVSR</code>)	Format	IV Offset (<code>PKTE_SA_CMD1.HSHCOFFST</code>)
DES/Triple-DES (CBC)	Previous Result of IV	Internal IV register [63:0]	0x00
	Input Buffer	Input buffer [63:0]	0x02
	Saved IV	State Record Saved IV [63:0]	0x00
	Automatic	PRNG output [63:0]	0x00
AES (CBC)	Previous Result of IV	Internal IV register [127:0]	0x00
	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	PRNG output [127:0]	0x00
AES (ICM) for Basic and SRTP	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
AES (CTR) for Basic and IP-sec	Input Buffer	SA_NONCE Input Buffer [63:0] 0x00000001	0x02
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	SA_NONCE PRNG output [95:32] 0x00000001	0x00

Notes:

1. The `PKTE_SA_CMD1.HSHCOFFST` bit field provides the IV offset. The offset is only applicable for basic hash or encrypt operations. For protocol operations, the offset is automatically enforced.

2. AES-CTR: The Nonce value as described in RFC 3686, is mapped to the `PKTE_SA_NONCE` register. This Nonce value remains constant for the lifetime of the security association.
3. The host processor controls the IV update using the save-IV bit (`PKTE_SA_CMD0.SVIV`). When part of a packet is processed using a stream cipher and the encrypt or decrypt data is not an integer multiple of the block size, the saved IV is invalid. It must not be used to resume processing the packet.
4. The packet engine supports automatic IV generation for outbound operations. A new IV is generated for every packet with the internal PRNG. This automatic IV generation can be used for all DES, Triple-DES, and AES modes that use an IV, except for AES-ICM mode.
5. For outbound operations, automatic IV generation is the most efficient. No additional I/O is required, and the host processor does not need to provide the IV. When the saved IV option supplies the IV, the IV must be changed for each packet sent. This activity happens when the packet engine writes back the IV to the state record after processing.
6. Outbound IPsec operations put the IV explicitly at the front of the packet. For an inbound IPsec operation, loading from the input buffer is most efficient and always used.
7. CBC processing must not use a predictable IV. Do not use the saved IV and previous result IV options for CBC processing. Refer to RFC 3602 for more details.

ARC4 Processing

The ARC4 algorithm supports two modes of operation: stateless and stateful. For SSL, TLS and DTLS operations that use the ARC4 algorithm, the mode must be set to stateful.

Stateless Mode. Each packet is processed with a newly initialized ARC4 key taken from the key field of the SA record. In this mode, the state information from the SA is never to be read.

CAUTION: If an ARC4 operation in stateless mode is interrupted by an *Interface Error* and the ARC4 state building process is aborted, the next packet fails. The ARC4 is not reset between two packets. For the next packet the ARC4 proceeds from the internal state that it was aborted. Even a soft-reset does not reset the ARC4 internal state, a hardware reset is required.

Stateful Mode. When the `PKTE_CTL_STAT.INITARC4` bit =1, the ARC4 algorithm initializes using the key specified in the SA record. When the `PKTE_CTL_STAT.INITARC4` bit =0, the ARC4 context is read from the ARC4 state field of the SA record and the *i* and *j* pointer field of the SA record. The encrypt and decrypt processing continues from this algorithm state.

The packet engine supports ARC4 key sizes of 40 bits to 128 bits. Longer keys can be used, but they cannot be made inside the packet engine.

The host processor applies ARC4 key scheduling function to the s-box and puts the 256-byte result into the ARC4 state record. It writes the *i* and *j* pointers in the SA (initial *i* = 1, *j* = 0). The host programs the packet engine and specifies stateful operation to continue the ARC4 algorithm.

Hash State Loading

The hash state can be loaded from various sources, depending on the selected protocol and hash algorithm. The *Different Sources for Loading the Hash State* table provides a list of all the options.

Table 33-16: Different Sources for Loading the Hash State

Hash Algorithm PKTE_SA_CMD0.HASHSRC =	From SA 0b00	RESERVED 0b01	From State 0b10	No Load 0b11
SHA-1	yes	-	yes	Yes
SHA-224	Yes	-	Yes	Yes
SHA-256	Yes	-	Yes	Yes
Null hash	x	x	x	x

Sequence Number Processing

The packet engine supports sequence number generation and verification for IPsec and extended SSL, TLS, and DTLS protocol operations.

A Sequence Number (SN) is an unsigned number that a sender must implement and a receiver can use to support anti-replay service (replay attacks) for a specific SA. This processing includes detection of the same packet that arrives more than once and detection of packets that arrive in an incorrect sequence and is outside an accepted level of order relative to the last received packet.

The packet engine supports the following options.

- Sequence number loaded from SA for outbound operations and is retrieved from the input packet for inbound operations.

The sequence number and sequence number mask fields are part of the SA record.

Sequence Number Processing in Extended SSL/TLS

SSL and TLS use an implicit sequence number of 64 bits that is not sent in the packet but part of the hash.

For inbound operations, no sequence number verification is performed; instead an incorrect sequence number results in an authentication error (PKTE_CTL_STAT.AUTHERR).

For outbound operations, with PKTE_SA_CMD0.HDRPROC enabled and PKTE_SA_CMD1.ENSQNCHK enabled, the packet engine generates a sequence number error when the 64-bit sequence number counter overflows (counter is $2^{64}-1$ and increments to 0). The host processor must not send the packet.

For both outbound and inbound operations, with PKTE_SA_CMD0.HDRPROC enabled and PKTE_SA_CMD1.ENSQNCHK enabled, the packet engine reads the sequence number from the PKTE_SA_SEQNUM[n] registers in the SA. When the operation is finished, the packet engine stores the incremented sequence number in the same SA fields. The sequence number mask PKTE_SA_SEQNUM_MSK[n] registers in the SA are not used.

For both outbound and inbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled, the packet engine does not increment the sequence number. It expects the host to provide the sequence number through the input buffer as part of the header.

Sequence Number Processing in DTLS

DTLS uses an explicit sequence number of 64 bits that is sent in the packet. The DTLS sequence number is composed of the epoch (16 bits) and packet sequence number (48 bits) that together form the 64-bit number, like the TLS sequence number. The epoch is incremented after each *Change Cipher Spec* message. The packet sequence number is incremented per packet starting from zero after each change cipher spec message.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled, the packet engine reads the sequence number from the `PKTE_SA_SEQNUM[n]` fields in the SA, then inserts the sequence number in the packet. Then the packet engine stores the incremented sequence number in the `PKTE_SA_SEQNUM[n]` fields in the SA. The sequence number mask `PKTE_SA_SEQNUM_MSK[n]` fields in the SA are not used.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine generates a sequence number error when the 48-bit sequence number counter overflows (counter is $2^{48} - 1$ and increments to 0). The host must not send the packet.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled or `PKTE_SA_CMD1.ENSQNCHK` disabled, the packet engine does not increment and verify a sequence number counter overflow, and therefore never generates a sequence number error. With `PKTE_SA_CMD0.HDRPROC` disabled the Packet Engine does not update the sequence number and sequence number mask fields in the SA and expects the host to provide the sequence number through the input buffer as part of the header.

For inbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine verifies the `PKTE_SA_SEQNUM[n]` fields against the sequence number in the packet using the `PKTE_SA_SEQNUM_MSK[n]` from the SA. Three situations can occur:

1. If the received sequence number falls outside and above the 64-bit sequence number mask, the mask is shifted. The packet engine updates the SA with the received sequence number and the shifted sequence number mask.
2. If the received sequence number falls inside the 64-bit sequence number mask and is not a duplicate sequence number (the same as received before), the corresponding bit in the mask is set. The packet engine updates the SA with the received sequence number and the updated sequence number mask.
3. If the received sequence number falls outside the 64-bit sequence number mask or matches an earlier received number a sequence number error is generated. The packet engine does not update the sequence number and sequence number mask fields in the SA. The host must discard the packet.

For inbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled or `PKTE_SA_CMD1.ENSQNCHK` disabled, the packet engine does not verify the sequence number against the sequence number in the packet and therefore never generates a sequence number error.

With `PKTE_SA_CMD0.HDRPROC` disabled the packet engine does not update the sequence number and sequence number mask fields in the SA and expects the host to provide the sequence number through the input buffer as part of the header.

The following tables provide details of sequence number processing.

Table 33-17: Sequence Number Generation and Verification Control (Outbound)

Header Processing SA_CMD0[19]	Anti-Replay Service SA_CMD1[29]	Description
Outbound		
1	1	Sequence number generation (increment)
1	1	Sequence number overflow check ($2^{32}-1$) to zero for IPsec, ($2^{48}-1$) to zero for DTLS and ($2^{64}-1$) to zero for Extended SSL/TLS.
1	0/1	Sequence number update in SA
0/1	0/1	Extended sequence number update in SA

Table 33-18: Sequence Number Generation and Verification Control (Inbound)

Header Processing SA_CMD0[19]	Anti-Replay Service SA_CMD1[29]	Description
Inbound		
1	1	Sequence number verification (check against sequence number and sequence number mask in SA). Applicable only for IPsec and DTLS.
1	0/1	Sequence number update in SA, except on authentication error
1	1	Sequence number mask update in SA. Applicable only for IPsec and DTLS.
0/1	0/1	Extended sequence number update in SA

Table 33-19: Header Processing Enabled, Anti-replay Service Enabled

	IPsec ESP	Ext. SSL	Ext. TLS	DTLS
Header processing enabled, anti-replay service enabled				
Inbound				
Initial value in the SA	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK	not used	not used	not used	not used
Value in the first packet or hashbyte stream, highest byte is B0	B0 B3 0x00000000 B4 B7 0x00000001	B0 B3 0x00000000 B4 B7 0x00000000	B0 B3 0x00000000 B4 B7 0x00000001	B0 B3 0xAA550000 B4 B7 0x00000001

Table 33-19: Header Processing Enabled, Anti-replay Service Enabled (Continued)

	IPsec ESP	Ext. SSL	Ext. TLS	DTLS
Header processing enabled, anti-replay service enabled				
Inbound				
Initial value in the SA after first packet	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM0[31:0]	0x00000001	0x00000001	0x00000001	0x00000001
SA_SEQNUM_MSK	not used	not used	not used	not used
Initial value in the SA	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0xFEDCBA98	0xFEDCBA98	0xFEDCBA98	0xAA55BA98
SA_SEQNUM0[31:0]	0x76543210	0x76543210	0x76543210	0x76543210
SA_SEQNUM_MSK	not used	not used	not used	not used
Value in the first packet or hashbyte stream, highest byte is B0	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xAA55BA98 B4 B7 0x76543211
Initial value in the SA after the packet	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM_1[31:0]	0xFEDCBA98	0xFEDCBA98	0xFEDCBA98	0xAA55BA98
SA_SEQNUM_0[31:0]	0x76543211	0x76543211	0x76543211	0x76543211
SA_SEQNUM_MASK	not used	not used	not used	not used
Highest value before overflow	B0 B3	B0 B3	B0 B3	B0 B3
SA_SEQNUM_1[31:0]	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xAA55FFFF
SA_SEQNUM_0[31:0]	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
SA_SEQNUM_MASK	not used	not used	not used	not used
Value in the packet or hashbyte stream, highest byte is B0	B0 B3 0x00000000B4 B7 0x00000000	B0 B3 0xFFFFFFFF B4 B7 0xFFFFFFFF	B0 B3 0xFFFFFFFF B4 B7 0xFFFFFFFF	B0 B3 0xAA55FFFF B4 B7 0xFFFFFFFF
Value after overflow in SA	B0 B3	B0 B3	B0 B3	B0 B3
SA_SEQNUM_1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM_0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MASK	not used	not used	not used	not used

Table 33-19: Header Processing Enabled, Anti-replay Service Enabled (Continued)

	IPsec ESP	Ext. SSL	Ext. TLS	DTLS
Header processing enabled, anti-replay service enabled				
Inbound				
Initial value in SA	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK1[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
Expected value in the first packet or hash-byte stream, highest byte is B0	B0 B3	B0 B3	B0 B3	B0 B3
	0x00000000	0x00000000	0x00000000	0xAA550000
	B4 B7	B4 B7	B4 B7	B4 B7
	0x00000001	0x00000001	0x00000001	0x00000001
Value in SA after first packet	B3 B0	B3 B0	B3 B0	B3 B0
	SA_SEQNUM1[31:0]	0x00000000	0x00000000	0xAA550000
	SA_SEQNUM0[31:0]	0x00000001	0x00000001	0x00000001
	SA_SEQNUM_MSK1[31:0]	0x00000000	0x00000000	0x00000000
	SA_SEQNUM_MSK0[31:0]	0x00000001	0x00000001	0x00000001

PKTE Block Diagram

The *PKTE Block Diagram* shows the functional blocks within the PKTE.

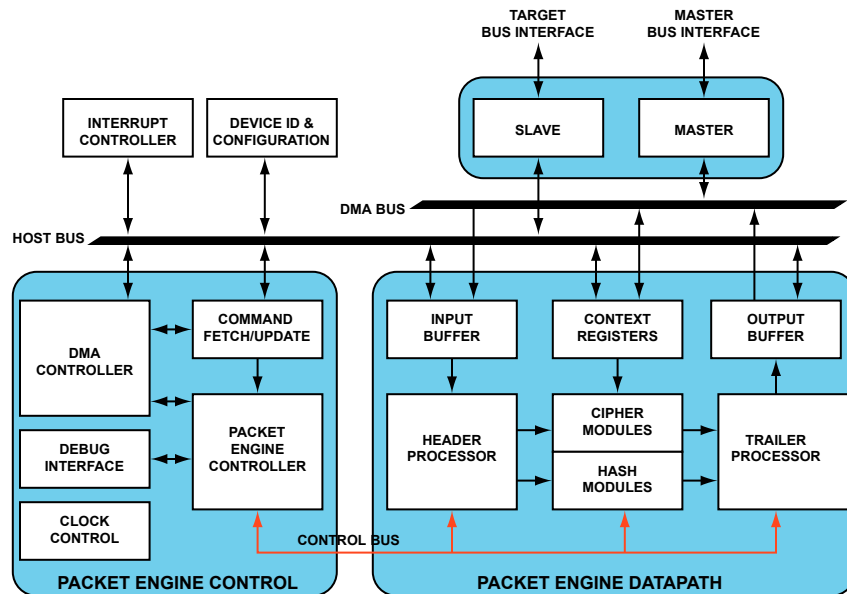


Figure 33-1: PKTE Block Diagram

PKTE Architectural Concepts

The following descriptions provide details on the functional blocks within the security packet engine.

Packet Engine

The packet engine contains symmetric cipher and hash engines. It is optimized to off-load intensive cryptographic operations from the host processor. It can perform parallel and pipelined encryption and hashing operations, reducing the latency, and processing time for packets that need both operations applied. The processor core provides the command information and packet data for the packet engine. The packet engine can run autonomously, using its local DMA controller to perform DMA transfers across the main bus to access the memory of the processor core. The DMA process incorporates flow-control to guarantee proper data flow. Two elements provide the command information that defines the processing for each packet:

- Command descriptor
- SA record

When the packet engine finishes the operation, it updates the SA record, when needed, and provides a result descriptor.

The packet engine has four different modes of operation. The modes give the processor core various levels of control over the command information and packet data transfers to and from the packet engine.

- Autonomous Ring Mode (ARM)
- Target Command Mode (with and without result descriptor ring) (TCM)
- Direct Host Mode (DHM)

Input/Output FIFO Buffers

The data for the packet engine is buffered at both input and output. These buffers decouple the DMA I/O process from the cipher and hash modules inside the packet engine. This functionality enables large DMA burst sizes and allows the crypto-engines to process data during I/O latency periods. Data moves automatically from the input buffer through the encryption and hash engines to the output buffer. If the output buffer is full, the process stops until the data is read and space is available in the output buffer. Each buffer is a 256-byte dual-port RAM.

Parallel Operations

The hash functionality and encrypt or decrypt functionality are tightly coupled. Operations that combine both hash and encrypt or decrypt functions are available to reduce processing time for data that must apply both. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of the hashing and the start of the encryption to support protocols such as IPSec and SRTP.

DMA Controller

The packet engine uses a high-performance DMA controller for autonomous data transfers for:

- Command descriptor reads
- SA record and state record reads
- Packet data read
- Result packet writes
- SA record and state record writes
- Result descriptor writes

Interrupt Controller

The packet engine includes an interrupt controller that, under programmable configuration control, can generate an interrupt on completion of certain operations. Individual interrupts can be masked and cleared. The interrupt registers show both the raw and masked interrupt status of the internal interrupts. The processor core can use interrupts, together with their associated threshold settings, to optimize the overall packet processing in the system. One interrupt can inform the processor core that the input side of the packet engine is almost empty to avoid a stall-on-empty condition. One interrupt can inform the host that the output side is almost full to avoid a stall-on-full condition. The controller uses several interrupts to inform the processor core about errors inside the packet engine. All available interrupts are combined into a single output port as either a level- or edge-active programmable interrupt output.

Clock Controller

The packet engine includes a clock controller that generates clock enable signals. A clock manager external to the packet engine uses the clock enable signals to switch the clocks to modules in the packet engine, reducing power consumption. The power saving can be significant, depending on the crypto-operation and the idle time of the packet engine. The clock controller generates the clock enable signals dynamically depending on the current crypto-operation. A clock control register provides the processor core the possibility to override this dynamic process.

PKTE Operating Modes

The packet engine can be configured in one of three command modes. For all modes the packet engine can generate an interrupt at completion of packet processing:

- **Autonomous Ring Mode (ARM)**. The core prepares descriptors in the CDR and then initiates a descriptor fetch by triggering the packet engine. When a packet operation is complete, the packet engine writes the result descriptor out into a ring in host memory using the system master bus interface.
- **Target Command Mode (TCM)**. The core directly writes the command descriptors to the packet command register set to initiate a packet operation. This process eliminates an extra DMA transfer to fetch a descriptor, but requires the core to synchronously initiate packet processing. This mode can be configured both without RDR or with RDR. In the latter case, the packet engine writes the result descriptor out into the RDR in host memory using the system master bus interface.
- **Direct Host Mode (DHM)**. The core has full control over the packet engine and uses the system slave bus interface. The core provides all the command descriptors, SA record and state record, and packet input data.

When the packet engine completes processing, the core must read the packet output data, the SA record, the state record, and the result descriptors.

Autonomous Ring Mode (ARM)

The *Autonomous Ring Mode* allows the packet engine and the host processor to operate asynchronously. A queue of multiple packets in the host processor memory can be processed continuously to provide the highest possible throughput. The packet engine autonomously fetches the command descriptor, the SA record, and optionally the state record and the input data from host processor memory. After the packet engine finishes processing, it autonomously writes the output data, updates the SA record and state record and writes the result descriptor in the host processor memory. It accesses the host processor memory through DMA read transfers across the system bus master.

This mode uses both command descriptor ring (CDR) and result descriptor ring (RDR).

Physically the CDR, RDR, SA record, source packet, and result packet can all be in different memories depending on the system memory architecture. The host processor writes command descriptors to the CDR in host processor memory. Then, it writes to the `PKTE_CDSC_INCR` register with the number of command descriptors that it prepared in the CDR. This write to the `PKTE_CDSC_INCR` register is the trigger for the packet engine to fetch the command descriptors sequentially from the CDR. When a command descriptor is fetched and written to the internal packet command register set, the descriptor is validated. If the ownership bits are set for the packet engine and the command is valid, processing starts. If not, that command is discarded and a result descriptor is written to the RDR with the error code *invalid command descriptor*. In this mode, the host processor can set a threshold on the CDR and enable an associated interrupt. The packet engine generates an interrupt when the number of command descriptors in the CDR is equal or below the threshold value.

The SA record and state record that contains the crypto context information are stored in a memory area. The packet engine autonomously accesses the memory area through DMA transfers across the system bus master. Also, the source packet and result packet are stored in a memory area that the packet engine autonomously accesses through the same bus master interface.

After decoding the command descriptor, the packet engine fetches the SA record and then, optionally, the state record.

Then, the source packet is fetched and stored in the input buffer. Packets less than the size of the input buffers are fetched entirely at once. Larger packets are fetched in parts that completely fill the input buffer. The packet engine initiates a new fetch each time the number of empty spaces in the input buffer reaches its threshold value. When the first packet data is available in the input buffer, the crypto engines start processing the data. After processing, the crypto engines write the result packet to the output buffer.

The packet engine writes the result packet from the output buffer to host processor memory when the number of bytes in the output buffer reaches its threshold value. Packets less than the threshold value are written entirely at once. Larger packets are written in parts that completely empty the output buffer.

The source packet data fetching, data processing, and result packet data writing are parallel processes that continue until the last result packet is written to host processor memory. Then, the packet engine optionally writes the SA record and the state record to update the crypto context information. As a final step, the packet engine writes the

result descriptor to the RDR. The host processor must either poll the RDR or wait for an interrupt from the packet engine to determine when packet processing is complete.

Target Command Mode (TCM)

This mode provides a synchronous interface between the processor core and the packet engine. The Command Descriptor Ring (CDR) is disabled and the processor core initiates packet processing by writing the command descriptor directly to the internal command descriptor MMRs of the packet engine. The Result Descriptor Ring (RDR) is optional.

- For `PKTE_CFG.MODE = 01`, the RDR is disabled and the processor core reads the result descriptor directly from the internal result descriptor register set for the packet engine.
- For `PKTE_CFG.MODE = 10`, the RDR is enabled and stored in a memory area that the packet engine can access through its master bus interface as in autonomous ring mode.

In target command mode, the packet engine autonomously fetches the SA record, state record, source packet data as in autonomous ring mode. Also, as in ARM, after processing, the packet engine updates state fields of the SA record and state record in the host processor memory.

Direct Host Mode (DHM)

This mode provides a synchronous interface between the processor core and the packet engine. The packet engine is under full control of the processor core. The host processor writes the command descriptors, SA record, and state record directly to the packet engine registers. Then, the processor core writes the source packet data into the input buffer. When processing is complete, the processor core reads back the result packet data from the output buffer. Finally, it reads the result descriptor, the updated SA record, and state record directly from the packet engine registers.

PKTE Event Control

The following section provides information about interrupts in the PKTE module.

PKTE Interrupt Signals

The Packet Engine has an internal Interrupt Controller with 9 interrupt sources. There are 7 registers associated with the interrupt controller:

1. Interrupt Unmasked Status - `PKTE_IUMSK_STAT`
2. Interrupt Mask Status - `PKTE_IMSK_STAT`
3. Interrupt Clear Register - `PKTE_INT_CLR`
4. Interrupt Enable Register - `PKTE_INT_EN`
5. Interrupt Mask Disable - `PKTE_IMSK_DIS`
6. Interrupt Mask Enable - `PKTE_IMSK_EN`

7. Interrupt Configuration - `PKTE_INT_CFG`

The *Packet Engine Interrupt Controller Block Diagram* shows the blocks of the interrupt controller.

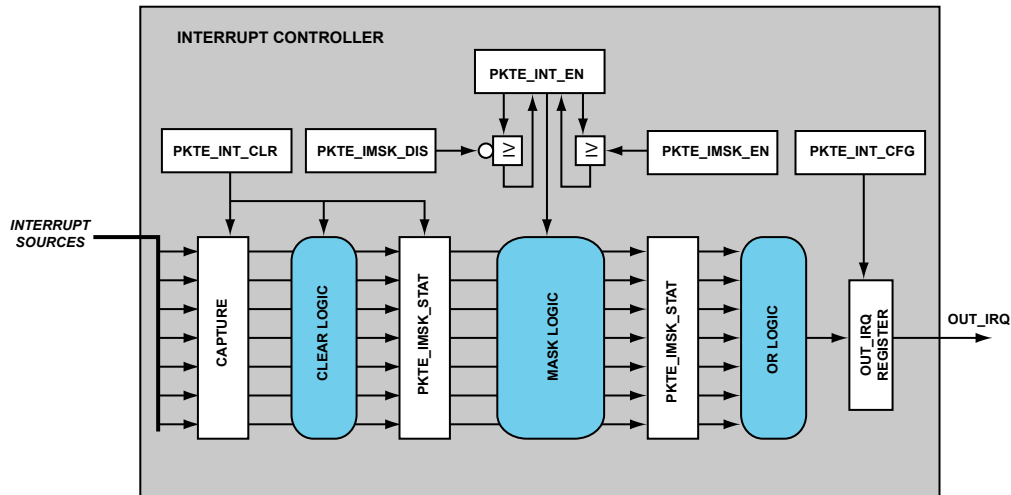


Figure 33-2: Packet Engine Interrupt Controller Block Diagram

All of the interrupt sources are pulse or level events in their native form.

These interrupts are captured and stored at their unmasked and masked status in their respective `PKTE_IUMSK_STAT` and `PKTE_IMSK_STAT` registers. This allows the host processor to read the status of any interrupt source either before or after the mask is applied.

The `PKTE_INT_EN` register provides a mask to select what interrupt source are enabled to the output interrupt request. Writing a one to the `PKTE_INT_CLR` register resets both the masked and unmasked interrupt.

The `PKTE_IMSK_EN` and `PKTE_IMSK_DIS` registers can be set to enable and disable individual interrupts respectively in the `PKTE_INT_EN` register. This avoids the need for read-modify-write operations from the host processor.

Ring Interrupts

Two interrupts are provided for efficient ring management: The CDR threshold interrupt (`cdrthrsh`) and the RDR threshold interrupt (`rdrthrsh`).

Command Descriptor Ring

The CDR threshold interrupt (`cdrthrsh`) is a level-based interrupt and connects to the threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit field. It enables the host processor to efficiently fill the CDR. The host processor writes command descriptors to the CDR with this interrupt masked until the CDR is full. Then the host processor enables the CDR threshold interrupt. When the interrupt is activated, the host processor clears the interrupt and it is guaranteed that it can put CDR threshold number of descriptors in the CDR.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.CDRTHRSH=224,
PKTE_INT_EN.CDRTHRSH=0 /*(IRQ disabled)*/
```

1. The host writes 8 Command Descriptors at once, then writes the `PKTE_CDSC_INCR` register to 8.
2. This is repeated until there are less than 8 empty entries in the CDR.
3. The fill level is now equal to the threshold (224)
4. The host enables the CDR threshold IRQ (`PKTE_INT_EN.CDRTHRSH=1`)
5. The packet engine processes packets and the fill level (`PKTE_CDSC_CNT` register) decreases.
6. Then the CDR threshold IRQ is activated as the fill level equals 224.
7. The host handles the interrupt, clears it and continues with step 1.

Result Descriptor Ring

The RDR threshold interrupt (`rdthrsh`) is a level-based interrupt and connects to the threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit field and the timeout value in the RD timeout (`PKTE_RING_THRESH.RDTO`) bit field. It enables the host processor to efficiently empty the RDR. The timeout reminds the host processor that when the threshold kicks in result descriptors stay long in the RDR and must be processed to reduce latency. The timeout counts when the ring is not empty, regardless of the fill level and restarts when the host processor writes the `PKTE_RDSC_CNT` register. Initially the host processor enables the RDR threshold interrupt. When the interrupt is activated the host processor reads result descriptors until the RDR is empty or contains less than the `PKTE_RING_THRESH.RDRTHRSH` number of descriptors.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.RDRTHRSH=32, timeout=1ms
PKTE_INT_EN.RDRTHRSH=1 /*(IRQ enabled)*/
```

1. The Packet Engine writes the Result Descriptor, timeout counter starts.
2. The Packet Engine writes 32 more Result Descriptors, fill level (`PKTE_RDSC_CNT` register) increases to 33.
3. The fill level exceeds threshold within 1ms, the RDR threshold IRQ is activated.
4. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. The write to the `PKTE_RDSC_DECR` register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.
5. The Packet Engine writes 8 more Result Descriptors, fill level increases to 33.
6. The fill level exceeds threshold within 1 ms, the RDR threshold IRQ is activated.
7. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. The write to the `PKTE_RDSC_DECR` register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.

8. After 1 ms, the timeout counter interrupt is activated.
9. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. This is repeated until there are less than 8 full entries in the RDR. The fill level is now under the threshold and the RDR threshold IRQ interrupt is inactive. Each write to the `PKTE_RDSC_DECR` register restarts the timeout counter.

PKTE Programming Model

The host processor must always follow a pre-defined sequence of five phases required by the packet engine on a per packet basis when using direct host mode. The following sections describe the five phases.

Phase 1. Write the Command Descriptor

1. Write the first command descriptor word with status and control information to the `PKTE_CTL_STAT` register.
2. Optionally, write the user ID to the `PKTE_USERID` register.
3. Write the last descriptor word to the `PKTE_LEN` register.
4. Write the value 0x1 to the `PKTE_CDSC_CNT` register. This operation triggers the packet engine to validate the command descriptor. If the command descriptor is invalid, an error is generated. (See the `PKTE_CTL_STAT` section in the Register Descriptions). If the command descriptor is valid, the packet engine waits for an `PKTE_SA_RDY` register write.

Phase 2. Write the State Registers, (ARC4 Buffer) and SA Registers

All required fields of the SA record and state record must be written. The fields required depend on the operation. The last field to be written is the `PKTE_SA_RDY` register. This register triggers the packet engine to start processing.

1. Write the required state record fields.
 - ARC4 state
 - IV
 - Digest count
 - State digest
2. Write the required SA record data.
3. To complete the SA record and state record, write the `PKTE_SA_RDY` register.

Phase 3. Write the Source Packet Data and Read Result Packet Data

The packet engine has input and output buffers. If a source packet is smaller than the size of the input buffer, then the packet can be written in one part. Otherwise, it must be written in multiple parts. The same applies to the

output data. If the result packet size is smaller than the size of the output buffer, then the packet can be read in one part. Otherwise, it must be read in multiple parts.

NOTE: An outbound packet that is smaller than the size of the input buffer can increase in size due to padding and does not always fit in the output buffer. Conversely, an inbound packet that is larger than the size of the input buffer can decrease in size, and due to de-padding, can fit in the output buffer. If the input buffer becomes empty or the output buffer becomes full, the engine stalls.

Two following steps describe different situations:

- Source packet smaller than the size of the input buffer, start at step 1.
- Source packet larger than the size of the input buffer, start at step 3.

The host processor must follow these steps:

1. Write the source packet data. Write the full source packet to the input buffer. Go to step 4.
2. Write the input buffer count register (`PKTE_INBUF_CNT`) with the number of valid bytes that are written to the input buffer. This value must correspond to the value in the `PKTE_LEN.TOTLEN` field of the command descriptor rounded up to the next multiple of 4 bytes. Go to step 5 to check the packet engine status.
3. Write part of the source packet data. The `PKTE_STAT.IBUFEMPTYCNT` field indicates the amount of free space in the input buffer. Programs write the number of bytes determined by the setting in the `PKTE_BUF_THRESH` register. Write the (partial) source packet to the input buffer. The host processor must resume where it ended the previous write operation. Do not write more than the buffer size at once. Go to step 4.
4. Write the `PKTE_INBUF_CNT` register with the number of valid bytes written to the input buffer. Go to step 5 to check the packet engine status.
5. Check packet engine status. Wait for an interrupt or poll the `PKTE_STAT` register for any of the following conditions:
 - Condition 1 - An error interrupt or any of the bits [7:5] in the `PKTE_STAT` register becomes active to indicate a packet processing error. Depending on the type of error, the host processor must take appropriate action. Usually, the result packet is not valid after a processing error has occurred. Go to phase 5 to read the result descriptor.
 - Condition 2 - An operation done interrupt or the `PKTE_STAT.OPDN` bit becomes active (the packet engine completed processing). Go to step 8 to read the remaining output data.
 - Condition 3 - An output buffer threshold interrupt or the `PKTE_STAT.OBUFREQ` bit becomes active. Go to step 6 to read a block of output data.
 - Condition 4 - An input buffer threshold interrupt or the `PKTE_STAT.IBUFREQ` bit becomes active. Go to step 3 to write a block of input data.
6. Read part of the output data. The `PKTE_STAT.OBUFFULLCNT` bit field indicates the number of bytes in the output buffer. This value is rounded up to full words. Programs read the number of bytes indicated in the

`PKTE_BUF_THRESH` register. Read the (partial) output packet from the output buffer. The host processor must resume where it ended the previous read operation. Do not read more than the input buffer size at once. Go to step 7.

7. Write the `PKTE_OUTBUF_CNT` register with the number of valid bytes read from the output buffer. Go to step 5 to check the packet engine status.
8. Read the remaining output data. The `PKTE_STAT.OBUFFULLCNT` bit field indicates the number of bytes in the output buffer. This value is rounded up to words. Read the (partial) packet output data from the output buffer. The host processor must resume where it ended the previous read operation. Go to step 9.
9. Write the `PKTE_OUTBUF_CNT` register with the number of valid bytes read from the output buffer. Go to phase 4.

Phase 4. Read the Result Descriptor

1. Read the first result descriptor word from the `PKTE_CTL_STAT`.
2. Optionally read the user ID from the `PKTE_USERID` register.
3. Read the last result descriptor word from the `PKTE_LEN` register.
4. Write the value 0x1 to the `PKTE_RDSC_DECR` register. This operation allows the packet engine to accept new command descriptors. Go to phase 5.

Phase 5. Read the SA Record and State Record

Depending on the operation, the SA record or state record is updated. Check the bit fields [23:16] in the `PKTE_CTL_STAT` register for the following conditions:

- Condition 1 - At least one error bit in the bit fields [23:16] of the `PKTE_CTL_STAT` register is set. Do not update the local host processor maintained version of the SA record and state record but take any required action.
- Condition 2 - None of the error bits in the bit fields [23:16] of the `PKTE_CTL_STAT` register is set, the packet is processed normally (without errors). Update the host processor maintained version of the SA record and state record with the result read from the packet engine registers:
 - ARC4 state
 - Sequence number
 - Sequence number mask
 - Result IV
 - Result digest count
 - Result digest

PKTE Mode Configuration

Before using the packet engine, it must be configured. The mode of the packet engine must be defined and the PRNG (if used) must be initialized.

Configure the packet engine in one of three command modes:

- Autonomous Ring Mode: `PKTE_CFG.MODE = b'11`
- Target Command Mode: `PKTE_CFG.MODE = b'10`
- Direct Host Mode: `PKTE_CFG.MODE = b'00`

PKTE Programming Concepts

The following sections provide conceptual information for programming the PKTE.

Packet Engine Descriptor

IMPORTANT: Depending on the mode, ARM, TCM, or DHM, the descriptor is either:

- in the memory of the host processor in the command descriptor ring, or
- written directly to the descriptor registers in the packet engine

References to descriptor registers are for either the register that is mirrored in the descriptor structure in memory or for the actual register itself.

Command descriptors are host-supplied commands that control the real-time operation of the packet engine. The packet engine returns result descriptors at the end of an operation that provide the status information to the host. The *Command Descriptor Structure* and the *Result Descriptor Structure* tables show these descriptors.

Table 33-20: Command Descriptor Structure

Word Offset	31:24	23:20	19:16	15:8	7:0	Address Offset
0	Pad Control	—		Next Header/ Pad Value	Control	0x000
1	Source Address					0x004
2	Destination Address					0x008
3	SA Address					0x00C
4	SA State Address					0x010
5	Reserved/ARC4 State Address					0x014
6	User ID					0x018
7	Bypass (words)	Control	Reserved	Input Packet Length (bytes)		0x01C

Table 33-21: Result Descriptor Structure

Word Offset	31:24	23:20	19:16	15:8	7:0	Address Offset
0	Pad Status	Status		Next Header/ Pad Value	Control	0x000
1	Source Address					0x004
2	Destination Address					0x008
3	SA Address					0x00C
4	SA State Address					0x010
5	Reserved/ARC4 State Address					0x014
6	User ID					0x018
7	Bypass (words)	Control	Reserved	Input Packet Length (bytes)		0x01C

When the packet engine is configured for autonomous ring mode, command descriptors and result descriptors reside in a ring in host memory. Command descriptors are automatically fetched from the Command Descriptor Ring (CDR) through DMA into the command descriptor registers. When an operation is complete, the result descriptors are automatically read from the packet engine and through DMA to the Result Descriptor Ring (RDR).

When the packet engine is configured for direct host mode, the host processor manually writes the command descriptor directly to the internal command descriptor MMR set. When an operation is complete, the host processor manually reads the result descriptor directly from the result descriptor MMR set.

The target command mode is a combination of the direct host mode and the autonomous ring mode. The host processor writes the command descriptor directly to the internal command descriptor register set. When an operation is complete there are two options.

1. The host processor can read the result descriptor directly from the result descriptor registers.
2. The result descriptors reside in a ring in host memory and the result descriptors are automatically DMA'd from the packet engine to the RDR.

When the host processor writes a command descriptor to the command descriptor registers, the packet engine is triggered when the host processor updates the `PKTE_CDSC_CNT` register. This functionality guarantees that all fields in the command descriptor are valid before the command is executed.

Descriptor Processing

This section describes the functional steps of the packet engine while processing the command descriptors.

Descriptor Ring Configuration

At initialization, the host processor specifies the size of the Command Descriptor Ring (CDR). The Result Descriptor Ring (RDR) has the same size.

NOTE: In some configurations, these two rings overlay each other with the results written on top of the command descriptors. This configuration is called overlaid ring mode.

When the packet engine is configured and enabled, it fetches the descriptors from the CDR using system bus master reads.

Descriptor Ring Processing

To validate the descriptor exchange between the host processor and the packet engine, the ownership bits `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` are used. One pair of ownership bits is in the first word of the descriptor (`PKTE_CTL_STAT`), and one pair is in the last word (`PKTE_LEN`). The ‘consumer’ of a descriptor must verify that both ownership pairs match to ensure that a race condition did not occur between one party writing and the other party reading the descriptor. A race condition can occur when a memory locking scheme is not used.

Each pair [`PKTE_CTL_STAT.PERDY`, `PKTE_CTL_STAT.HOSTRDY`] of ownership bits provide 3 states:

- b'00 = idle or null descriptor
- b'01 = host processor has written a descriptor in the CDR and passed ownership to the packet engine
- b'10 = packet engine processing complete: packet engine has written the descriptor in the RDR and passed ownership back to the host.
- b'11 = Reserved

At initialization, the host sets the entire CDR memory area to zero, when the CDR is used.

1. The host processor writes one or more command descriptors to the CDR. The host processor must set the `PKTE_CTL_STAT.HOSTRDY` bit to 1 and the `PKTE_CTL_STAT.PERDY` bit to 0 to indicate that ownership has passed to the packet engine. These bits are mirrored in the `PKTE_LEN` descriptor word.
2. The host processor must write the `PKTE_CDSC_INCR` register with the number of new valid command descriptors in the CDR.
3. The packet engine reads and validates one command descriptor.
4. The packet engine reads the SA record and state record, processes the packet and updates the SA record and state record.
5. If the rings are not overlaid and the `PKTE_CFG.ENCDRUPDT` bit is 1, the packet engine writes the result descriptor to the CDR with the `PKTE_CTL_STAT.HOSTRDY` bit set to 0 and `PKTE_CTL_STAT.PERDY` bit set to 1. These bits are mirrored in the `PKTE_LEN` descriptor word.
6. The packet engine writes the result descriptor to the RDR. The packet engine sets the `PKTE_CTL_STAT.PERDY` bit to 1 and the `PKTE_CTL_STAT.HOSTRDY` bit to 0 to indicate that the ownership has passed to the host. These bits are mirrored in the `PKTE_LEN` descriptor word.
7. The packet engine decrements the value in the `PKTE_CDSC_CNT` register. If the value is not zero, the packet engine reads with the next command descriptor (step 3).

8. The packet engine increments the value in the `PKTE_RDSC_CNT` register.
9. The host processor reads one or more result descriptors from the RDR and processes the results.
10. The host processor must write the `PKTE_RDSC_DECR` register with the number of processed result descriptors in the RDR.

Descriptor Ownership

The ownership of the command descriptor and result descriptor is set by ownership bits in the first and last word of the respective descriptor, in the `PKTE_CTL_STAT` register and the `PKTE_LEN` register. For the command descriptor, it is the processor core that sets the ownership to the packet engine. For the result descriptor, it is the packet engine that sets the ownership to the host processor.

The packet engine reads the ownership bits before processing, irrespective of the mode of the packet engine. The ownership bits are used to validate and identify the descriptor. When two separate rings are used, the packet engine can be programmed to clear the ownership bits of the command descriptors in the CDR so the host processor knows which descriptors are processed.

NOTE: This update of the ownership bits can be disabled in the `PKTE_CFG` register when the host processor actively counts the number of descriptors in the CDR to prevent ring wrapping.

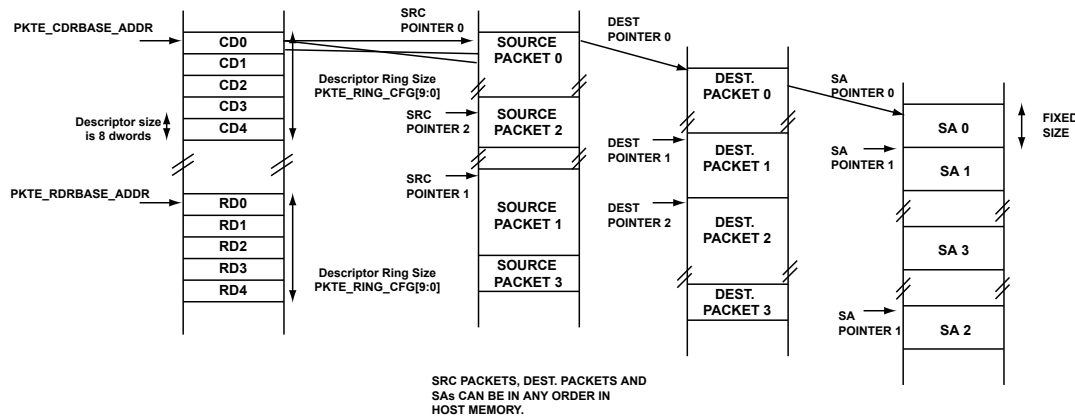


Figure 33-3: Descriptor Rings in Autonomous Ring Mode

SA Record and State Record Structure

The SA record is a packed structure that contains the remainder of the information needed by the packet engine to process a packet. Most of the information fields in the SA record, such as the key and encryption mode, are static for the lifetime of the association. The fields do not require frequent manipulation by the host processor. The SA record non-static fields are the sequence number and sequence number mask.

The SA record can have a corresponding state record that is used to save results from the current operations that can be used for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

If an SA record is used for operations that use ARC4 processing in stateful mode, it has a corresponding ARC4 state record that holds the ARC4 State.

In this manual, the state record and the ARC4 state record are referred to as state record.

There is no practical limit to how many SA records and corresponding state records the packet engine can support.

In the autonomous ring mode and target command mode, once the packet engine has validated a command descriptor, it automatically fetches the SA record and optional state record. After processing, the packet engine updates the stateful fields in the SA record and state record in the host processor memory.

In direct host mode, after the descriptor is validated, the host must write the SA record directly into the internal registers of the packet engine. After processing, the host reads the stateful fields from the SA registers of the packet engine and saves them back to the SA record in the host processor memory.

SA Record Structure

The *SA Record Structure* table shows the structure for an SA Record. When using direct host mode, the corresponding elements are accessed directly with the registers. When using autonomous ring mode or target command mode, the SA Record is defined, configured and accessed in host memory.

Table 33-22: SA Record Structure

Word Offset	Description (name)	Use
0	PKTE_SA_CMD0[31:0]	SA Control word 0 (all operations)
1	PKTE_SA_CMD1[31:0]	SA Control word 1 (all operations)
2	PKTE_SA_KEY0[31:0]	Key word (DES, Triple-DES, AES-128/192/256, ARC4,)
3	PKTE_SA_KEY1[63:32]	Key word (DES, Triple-DES, AES-128/192/256, ARC4,)
4	PKTE_SA_KEY2[95:64]	Key word (Triple-DES, AES-128/192/256, ARC4,)
5	PKTE_SA_KEY3[127:96]	Key word (Triple-DES, AES-128/192/256, ARC4,)
6	PKTE_SA_KEY4[159:128]	Key word (Triple-DES, AES-192/256)
7	PKTE_SA_KEY5[191:160]	Key word (Triple-DES, AES-192/256)
8	PKTE_SA_KEY6[223:192]	Key word (AES-256)
9	PKTE_SA_KEY7[255:224]	Key word (AES-256)
10	PKTE_SA_IDIGEST0[31:0]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
11	PKTE_SA_IDIGEST1[63:32]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
12	PKTE_SA_IDIGEST2[95:64]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
13	PKTE_SA_IDIGEST3[127:96]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)

Table 33-22: SA Record Structure (Continued)

Word Offset	Description (name)	Use
14	PKTE_SA_IDIGEST4[159:128]	Inner Hash digest (Basic Hash and HMAC with SHA-1, SHA-224, SHA-256)
15	PKTE_SA_IDIGEST5[191:160]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
16	PKTE_SA_IDIGEST6[223:192]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
17	PKTE_SA_IDIGEST7[255:224]	Inner Hash digest (Basic Hash and HMAC with SHA-256)
18	PKTE_SA_ODIGEST0[31:0]	Outer Hash digest (HMAC with MD5, SHA-1, SHA-224, SHA-256)
19	PKTE_SA_ODIGEST1[63:32]	Outer Hash digest (HMAC with MD5, SHA-1, SHA-224, SHA-256)
20	PKTE_SA_ODIGEST2[95:64]	Outer Hash digest (HMAC with MD5, SHA-1, SHA-224, SHA-256)
21	PKTE_SA_ODIGEST3[127:96]	Outer Hash digest (HMAC with MD5, SHA-1, SHA-224, SHA-256)
22	PKTE_SA_ODIGEST4[159:128]	Outer Hash digest (HMAC with SHA-1, SHA-224, SHA-256)
23	PKTE_SA_ODIGEST5[191:160]	Outer Hash digest (HMAC with SHA-224, SHA-256)
24	PKTE_SA_ODIGEST6[223:192]	Outer Hash digest (HMAC with SHA-224, SHA-256)
25	PKTE_SA_ODIGEST7[255:224]	Outer Hash digest (HMAC with SHA-256)
26	PKTE_SA_SPI[31:0]	SPI (IPsec), Type[23:16] / Version [15:0] (SSL, TLS, DTLS)
27	PKTE_SA_SEQNUM0[31:0]	Sequence Number (IPsec, SSL, TLS, DTLS with Header Processing)
28	PKTE_SA_SEQNUM1[63:32]	
29	PKTE_SA_SEQNUM_MSK0[31:0]	Sequence Number Mask (IPsec, DTLS inbound with Header Processing)
30	PKTE_SA_SEQNUM_MSK1[63:32]	
31	PKTE_SA_NONCE[31:0] / PKTE_SA_READY	Nonce value (AES-CTR, AES-ICM)/ARC4, i and j pointers (ARC4,)/SA ready indicator (Direct Host Mode)

Some of these fields may be updated by the packet engine. These include:

- PKTE_SA_SEQNUM0
- PKTE_SA_SEQNUM1
- PKTE_SA_SEQNUM_MSK0

- PKTE_SA_SEQNUM_MSK1

All the other fields remain unchanged.

SA State Structure

The security association state structure contains information that may be updated after each packet, such as the IV and the intermediate hash result. The *SA State Structure* table shows the SA state structure and usage. In direct host mode, the elements are accessed directly using the PKTE registers. In target command mode and autonomous ring mode, this structure is defined and updated in host memory.

Table 33-23: SA State Structure

Word Offset	Description (name)	Use
0	PKTE_STATE_IV0[31:0]	Initialization Vector (DES, Triple DES, AES)
1	PKTE_STATE_IV1[63:32]	Initialization Vector (DES, Triple DES, AES)
2	PKTE_STATE_IV2[95:64]	Initialization Vector (AES)
3	PKTE_STATE_IV3[127:96]	Initialization Vector (AES)
4	PKTE_STATE_BYTE_CNT0[31:0]	Current hash byte count (MD5, SHA-1, SHA-224, SHA-256)
5	PKTE_STATE_BYTE_CNT1[63:32]	Current hash byte count (MD5, SHA-1, SHA-224, SHA-256)
6	PKTE_STATE_IDIGEST0[31:0]	Inner Hash digest (mirror of PKTE_SA_IDIGEST0)
7	PKTE_STATE_IDIGEST1[63:32]	Inner Hash digest (mirror of PKTE_SA_IDIGEST1)
8	PKTE_STATE_IDIGEST2[95:64]	Inner Hash digest (mirror of PKTE_SA_IDIGEST2)
9	PKTE_STATE_IDIGEST3[127:96]	Inner Hash digest (mirror of PKTE_SA_IDIGEST3)
10	PKTE_STATE_IDIGEST4[159:128]	Inner Hash digest (mirror of PKTE_SA_IDIGEST4)
11	PKTE_STATE_IDIGEST5[191:160]	Inner Hash digest (mirror of PKTE_SA_IDIGEST5)
12	PKTE_STATE_IDIGEST6[223:192]	Inner Hash digest (mirror of PKTE_SA_IDIGEST6)
13	PKTE_STATE_IDIGEST7[255:224]	Inner Hash digest (mirror of PKTE_SA_IDIGEST7)

ARC4 State Structure

The *ARC4 State Structure* table describes the state structure used with ARC4. When using the PKTE in direct host mode, these fields are accessed with the registers, starting at the value in the `PKTE_ARC4STATE_BUF` register. When using the PKTE in autonomous ring mode or target command mode, these fields are defined and accessed in a structure in host memory.

Table 33-24: ARC4 State Structure

Word Offset	Description (name) ^{*1}	Use
0	PKTE_ARC4_STATE0[3:0] ^{*2}	ARC4 (Basic, SSL and TLS)
1	PKTE_ARC4_STATE1[7:4]	
...	...	
62	PKTE_ARC4_STATE2[251:248]	
63	PKTE_ARC4_STATE3[255:252]	

*1 The indices in these fields indicate bytes.

*2 There are no corresponding named registers for these fields. PKTE_ARC4_STATE0 corresponds to [PKTE_ARC4STATE_BUF](#) and PKTE_ARC4_STATE1 corresponds to the 32-bit register following [PKTE_ARC4STATE_BUF](#) and so on.

Configuring Operations in the PKTE

The operation (cipher, hash function, and others) that the PKTE performs is configured primarily in the [PKTE_SA_CMD0](#) register. The following sections include a series of tables to help configure the least significant 16 bits of the [PKTE_SA_CMD0](#) register. These fields include:

- The operation code field (`PKTE_SA_CMD0.OPCD`)
- The direction field (`PKTE_SA_CMD0.DIR`)
- The operation group field (`PKTE_SA_CMD0.OPGRP`)
- The padding type (`PKTE_SA_CMD0.PADTYPE`)
- The cipher selection (`PKTE_SA_CMD0.CIPHER`)
- The hash selection (`PKTE_SA_CMD0.HASH`)

Basic Operations and Decoding

Table 33-25: Basic Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b00	0	0b000	Encrypt	0b00	1	0b000	Decrypt
0b00	0	0b001	Encrypt - Hash	0b00	1	0b001	Hash - Decrypt
0b00	0	0b010	Reserved	0b00	1	0b010	Reserved
0b00	0	0b011	Hash	0b00	1	0b011	Hash
0b00	0	0b100... 0b110	Reserved	0b00	1	0b100... 0b110	Reserved
0b00	0	0b111	PRNG	0b00	1	0b111	Reserved

Table 33-26: Protocol Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b01	0	0b000	ESP Outbound	0b01	1	0b000	ESP Inbound
0b01	0	0b001... 0b011	Reserved	0b01	1	0b001	Reserved
0b01	0	0b100	Basic SSL Outbound	0b01	1	0b010	Basic SSL Inbound
0b01	0	0b101	Basic TLS Outbound	0b01	1	0b011	Basic TLS Inbound
0b01	0	0b110	Reserved	0b01	1	0b100... 0b110	Reserved
0b01	0	0b111	SRTP Outbound	0b01	1	0b111	SRTP Inbound

NOTE: For SSL/TLS and SRTP, no header processing is performed in hardware.

Table 33-27: Extended Protocol Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b11 0	0	0b000	Reserved	0b11	1	0b000	Reserved
0b11 0	0	0b001	DTLS Outbound	0b11	1	0b001	DTLS Inbound
0b11 0	0	0b010... 0b011	Reserved	0b11	1	0b010... 0b011	Reserved
0b11 0	0	0b100	Ext. SSL Outbound	0b11	1	0b100	Ext. SSL Inbound
0b11 0	0	0b101	Ext. TLS v1.0 Outbound	0b11	1	0b101	Ext. TLS v1.0 Inbound
0b11 0	0	0b110	Ext. TLS v1.1 Outbound	0b11	1	0b110	Ext. TLS v1.1 Inbound
0b11 0	0	0b111	Reserved	0b11	1	0b111	Reserved

Error Code Description

The `PKTE_CTL_STAT` register is used to configure the packet engine for processing in Direct Host Mode (DHM) or Target Command Mode (TCM). The `PKTE_CTL_STAT` structure element in memory is used when the packet

engine is configured for Autonomous Ring Mode (ARM). In both cases, when an operation is started, errors are reported in the status field (bits [23:16]) of this register or structure element. The *Extended Error Codes - Status Encoding* table provides a guide on how to decipher the meaning of the bits that are set when an error occurs.

Extended Error Codes

The following table provides information about the extended errors associated with the PKTE module.

Table 33-28: Extended Error Codes - Status Encoding

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0000_0000	0x00	NA	Successful completion. No errors occurred during processing of the packet.	Packet fully processed
0b----_---1	0x-1	NA	Authentication Error. For an inbound IPsec ESP operation, the Integrity Check Value (ICV) does not match the computed value. For an inbound SRTP operation, the authentication tag does not match the computed value. For a basic SSL/TLS, Extended SSL/TLS or DTLS operation the Message Authentication Code (MAC) does not match the computed value.	Packet fully processed
0b----_--1-	0x-2	NA	Pad Verify Error. For inbound operations that use pad type Constant TLS, IPsec or PKCS#7, the decrypted pad does not match the expected values for the selected pad type.	Packet fully processed
0b----_-1--	0x-4	NA	Sequence Number Error. For an inbound IPsec or DTLS operation, there was a fault in the Anti-Replay Sequence Number. For an outbound IPsec packet, the sequence number overflows; count is $2^{32}-1$ and increments to 0. For an outbound DTLS operation, the sequence number overflows; count is $2^{48}-1$ and increments to 0. For an outbound SSL or TLS operation, the sequence number overflows; count is $2^{64}-1$ and increments to 0.	Packet fully processed
0b0000_1---	0x08	1	System Bus error. The master bus interface generates an error due to ERROR response from system slave. The slave bus interface generates an error due to request for non-word (32-bit) access.	Packet is aborted. The host must reject the packet and apply a hardware reset to the system.

Table 33-28: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0001_1---	0x18	2	Invalid Command Descriptor Error. The ownership bits in the command descriptor are not set to the packet engine, after the <code>PKTE_CDSC_CNT</code> register is incremented.	Command descriptor is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0010_1---	0x28	3	Invalid Crypto Operation Error. A reserved operation is selected.	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0011_1---	0x38	4	Invalid Crypto Algorithm Error. A reserved cipher is selected, refer to <code>PKTE_SA_CMD0.CIPHER</code> . A reserved hash is selected, refer to <code>PKTE_SA_CMD0.HASH</code> .	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0100_1---	0x48	5	SPI Error. On an inbound packet, the 32-bit SPI value in the packet does not match the value in the SA while header processing is enabled. Note: A failure caused by an SPI mismatch, in general should not occur because the host checks the SPI and does not send an incorrect SPI to the packet engine.	Packet is fully processed. The host must reject the packet.
0b0101_1---	0x58	3	Zero Length Error. The packet length defined in the command descriptor <code>PKTE_LEN.TOTLEN</code> is zero, which is illegal.	Packet command is ignored, no packet processed. The host must reject the packet.

Table 33-28: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0110_1xxx	0x68	6	<p>Invalid Packet Length Error.</p> <p>For Basic Encrypt-Hash and Hash-Decrypt operations: $PKTE_LEN.TOTLEN < \text{Hash/Encrypt Offset}$</p> <p>For IPsec ESP inbound operations: $PKTE_LEN.TOTLEN < \text{ICV length}$ or $PKTE_LEN.TOTLEN$ is non-4 byte aligned</p> <p>For SRTP inbound operations: $PKTE_LEN.TOTLEN \leq \text{IV (opt.)} + \text{Bypass Offset} + \text{ROC}$</p> <p>For SSL inbound operations: $PKTE_LEN.TOTLEN \leq 1$ or packet length > 65535 bytes (SSL packet-bypass length)</p> <p>For TLS and DTLS inbound operations: $PKTE_LEN.TOTLEN \leq 13$ or payload length > 65535 bytes (data to be hashed)</p> <p>Note: For IPsec ESP the ICV is stripped before the length is checked.</p>	<p>Packet processing is aborted.</p> <p>Result packet length is zero.</p> <p>The host must reject the packet and apply a software reset of the packet engine.</p>
0b0111_1xxx	0x78	7	<p>Block Size Error.</p> <p>The length of the inbound packet defined in the Command Descriptor $PKTE_LEN.TOTLEN$ is not a multiple of the DES or AES block cipher length. For outbound packets the size is always automatically aligned (padded) to the correct block size. The hashed packet length is not a multiple of the hash block size for intermediate hash operation.</p> <p>For a final hash operation no error is generated.</p> <p>Note: For IPsec ESP operations the ICV is stripped before the block size is checked.</p>	<p>Packet is fully processed.</p> <p>The host must reject the packet.</p>
0b1000_1xxx	0x88	8	<p>Processing Error.</p> <p>The number of bytes in the input buffer is more than defined in the $PKTE_LEN.TOTLEN$ field. The number of bytes written to the output buffer is less than processed in the datapath.</p>	<p>Packet processing aborted.</p> <p>Result packet length is zero.</p> <p>The host must reject the packet and apply a software reset.</p>
0b1010_1xxx 0b1111_1xxx	Reserved			

Number Format

When dealing with cryptographic functions, data and keys are large vectors. For instance, AES supports keys of sizes 128, 192, and 256 bits. When a key needs to be loaded or read, multiple 32-bit key registers are used, namely

`PKTE_SA_KEY[n]` registers. The first key register `PKTE_SA_KEY[n]0` holds bits 31:0, and `PKTE_SA_KEY[n]1` holds the next thirty two bits 63:32, and so on.

Generally, for large vectors defined as Byte0, Byte1, Byte2, Byte3 and so on, the values are stored in the PKTE registers as `PKTE_REG0 = Byte3Byte2Byte1Byte0`, followed by `PKTE_REG1 = Byte7Byte6Byte5Byte4` and so on.

PKTE Programming Examples

Use these examples to extend your understanding of PKTE features, operating modes, event control, and programming modes.

Calculating SHA in Direct Host Mode

This section describes how to configure the packet engine to calculate a hash digest using one of supported SHA algorithms in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the `PKTE_CFG.MODE` bit =0
2. Set the ownership back to the packet engine to process a command descriptor by setting the `PKTE_CTL_STAT.PERDY` bit =0 and the `PKTE_CTL_STAT.HOSTRDY` bit =1.
3. Set the `PKTE_CTL_STAT.HASHFINAL` bit to indicate this command descriptor handles all the input data for the hash calculation. This configuration is needed for the packet engine because the last block requires special handling (see FIPS 180-4 for details).
4. Set size of the input data in bytes in the `PKTE_LEN.TOTLEN` bit field.
5. Also set the `PKTE_LEN.PEDONE` bit =0 and the `PKTE_LEN.HSTRDY` bit =1. These bits must be the same as the `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` bits to guarantee ownership.
6. Set the `PKTE_CDSC_CNT` register =1 to trigger the packet engine to start validating the command descriptor. In this case, the `PKTE_CTL_STAT`, `PKTE_LEN` and `PKTE_CDSC_CNT` registers are the only command descriptor registers modified.
7. Configure the `PKTE_SA_CMD0` and `PKTE_SA_CMD1` registers to define the operation. For an SHA, set the `PKTE_SA_CMD0.OPCD` bit field =0b011 for hash operation and the `PKTE_SA_CMD0.OPGRP` bit field =0b00 for basic operation.
8. Select the specific SHA function using the `PKTE_SA_CMD0.HASH` bit field as follows.
 - For SHA-1, `PKTE_SA_CMD0.HASH =0b0001`
 - SHA-224, `PKTE_SA_CMD0.HASH =0b0010`
 - for SHA-256, `PKTE_SA_CMD0.HASH =0b0011`
9. Depending on the SHA selected, the appropriate digest length must be chosen for the `PKTE_SA_CMD0.DIGESTLEN` bit field as follows.

- For SHA-1, `PKTE_SA_CMD0.DIGESTLEN = 0b0101` (5 words)
 - For SHA-224, `PKTE_SA_CMD0.DIGESTLEN = 0b0111` (7 words)
 - For SHA-256, `PKTE_SA_CMD0.DIGESTLEN = 0b1000` (8 words)
10. The SHA specifies initial constants. These constants can be pre-loaded or read from memory. In this example, by setting the `PKTE_SA_CMD0.HASHSRC` bit field = 0b11, the packet engine provides the correct initial constants depending on the SHA chosen.
 11. Next, set the `PKTE_SA_CMD1.CPYDGST` bit = 1 and `PKTE_SA_CMD1.CPYPAD` bit = 1 to move the result to the output buffer of the packet engine at the `PKTE_DATAIO_BUF` location.
 12. At this point, write to the `PKTE_SA_RDY` register with any value to trigger the operation.
 13. Start writing the input to the data buffer of the packet engine starting at the `PKTE_DATAIO_BUF` location.
 14. Write the `PKTE_INBUF_CNT` register with the length of the input rounded up to the next multiple of 4. For example, if the input length is 30 bytes, set this register to 32.
 15. Poll the `PKTE_STAT` register to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the digest is available in the packet engine data I/O buffer.

NOTE: The input data or message is input into the packet engine data buffer in big endian format while the result or digest is little endian format.

Performing AES Decryption in Direct Host Mode

This section describes how to configure the packet engine to decrypt using AES-128 in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the `PKTE_CFG.MODE` bit = 0
2. Start configuring the command descriptor registers. Set the ownership back to the packet engine to process a command descriptor by setting the `PKTE_CTL_STAT.PERDY` bit = 0 and the `PKTE_CTL_STAT.HOSTRDY` bit = 1.
3. Next, configure the `PKTE_LEN.TOTLEN` bit field with the size of the packet or message to decrypt. If the entire input message (cipher text) fits into the 256-byte data I/O buffer of the packet engine, the process can be done in one shot.
4. Set the `PKTE_LEN.PEDONE` bit = 0 and the `PKTE_LEN.HSTRDY` bit = 1. These bits must have the same setting as the `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` bits to guarantee ownership.
5. Set the `PKTE_CDSC_CNT` register = 1 to trigger the packet engine to start validating the command descriptor. In this case, the `PKTE_CTL_STAT`, `PKTE_LEN`, and `PKTE_CDSC_CNT` registers are the only command descriptor registers modified.
6. Next, configure the `PKTE_SA_CMD0` and `PKTE_SA_CMD1` registers to define the operation.

- For a AES decrypt inbound cipher operation, set the `PKTE_SA_CMD0.OPCD` bit field =0b000 and the `PKTE_SA_CMD0.DIR` bit field =0b1.
 - Set the `PKTE_SA_CMD0.OPGRP` bit field =0b00 for basic operation.
 - To choose the AES cipher, set the `PKTE_SA_CMD0.CIPHER` bit field =0b0011. Set the `PKTE_SA_CMD0.HASH` bit field to 0b1111 to choose the NULL function.
7. Next, set the `PKTE_SA_CMD1.AESKEYLEN` bit field to select the appropriate key length. In this case, setting it to 0b10 select 128 bits. Also, set `PKTE_SA_CMD1.CIPHERMD` bit field to select the mode. In this case, setting it to 0b01 select CBC mode.
 8. Continue configuring the Security Association (SA) record by loading the key in the `PKTE_SA_KEY[n]` registers.
 9. Next load the initialization vector in the SA state registers (`PKTE_STATE_IV[n]`).
 10. Finally, write anything in to the `PKTE_SA_RDY` register to trigger the operation.
 11. The input data can now be written into the data I/O buffer starting at `PKTE_DATAIO_BUF`.
 12. After the data is written, write the length (or next multiple of 4) into `PKTE_INBUF_CNT` register.
 13. Poll `PKTE_STAT` to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the result can be found in the same data I/O buffer.

ADSP-2156x PKTE Register Descriptions

Security Packet Engine (PKTE) contains the following registers.

Table 33-29: ADSP-2156x PKTE Register List

Name	Description
<code>PKTE_ARC4STATE_ADDR</code>	Packet Engine ARC4 State Record Address
<code>PKTE_ARC4STATE_BUF</code>	Starting Entry of 256-byte ARC4 State Buffer
<code>PKTE_BUF_PTR</code>	Packet Engine Buffer Pointer Register
<code>PKTE_BUF_THRESH</code>	Packet Engine Buffer Threshold Register
<code>PKTE_CDRBASE_ADDR</code>	Packet Engine Command Descriptor Ring Base Address
<code>PKTE_CDSC_CNT</code>	Packet Engine Command Descriptor Count Register
<code>PKTE_CDSC_INCR</code>	Packet Engine Command Descriptor Count Increment Register
<code>PKTE_CFG</code>	Packet Engine Configuration Register
<code>PKTE_CLK_CTL</code>	PE Clock Control Register
<code>PKTE_CONT</code>	PKTE Continue Register
<code>PKTE_CTL_STAT</code>	Packet Engine Control Register

Table 33-29: ADSP-2156x PKTE Register List (Continued)

Name	Description
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_ARC4IJPTR	ARC4 i and j Pointer Register
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers

Table 33-29: ADSP-2156x PKTE Register List (Continued)

Name	Description
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

Packet Engine ARC4 State Record Address

The `PKTE_ARC4STATE_ADDR` register holds the start address of the SA ARC4 state record.

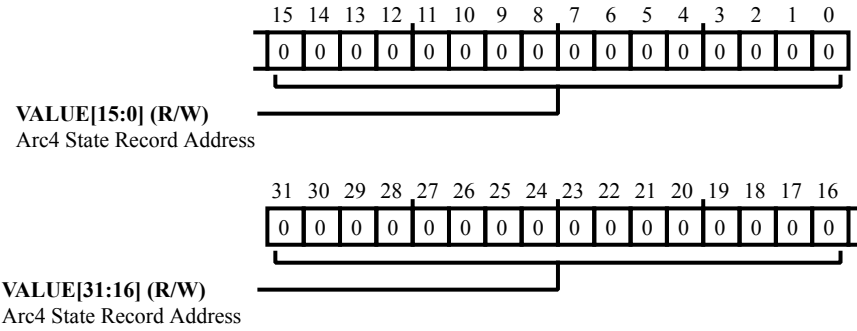


Figure 33-4: `PKTE_ARC4STATE_ADDR` Register Diagram

Table 33-30: `PKTE_ARC4STATE_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Arc4 State Record Address.

Starting Entry of 256-byte ARC4 State Buffer

The `PKTE_ARC4STATE_BUF` register is used to store the pre-processed key that initializes the ARC4 module. In direct host mode, before processing starts, the Host must write the ARC4 state, starting from the base address and increment the address pointer for each write. When processing completes the Host must read the ARC4 state and copy it to the local Host maintained state record. After a reset, a read from any address in the address range of the ARC4 buffer returns an undefined value.

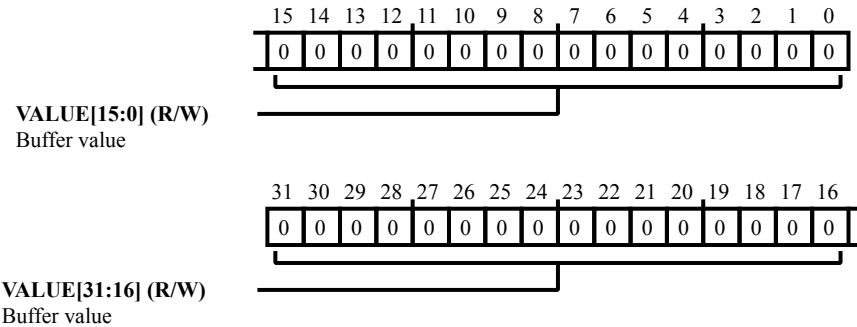


Figure 33-5: `PKTE_ARC4STATE_BUF` Register Diagram

Table 33-31: `PKTE_ARC4STATE_BUF` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Buffer value. The <code>PKTE_ARC4STATE_BUF.VALUE</code> bit field stores the pre-processed key that initializes the ARC4 module.

Packet Engine Buffer Pointer Register

The `PKTE_BUF_PTR` register contains the offset of the next buffer address (entry) to be read or written by the packet engine. This register is used in direct host mode only.

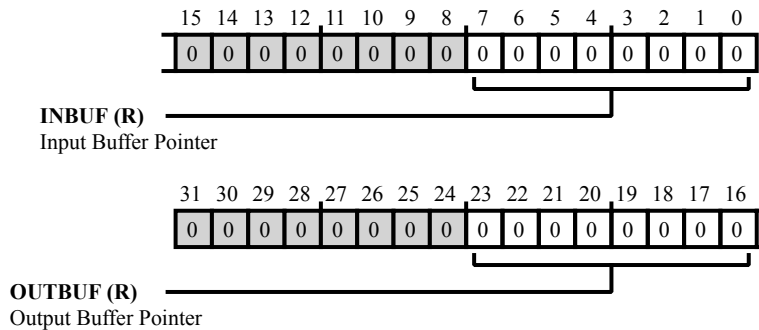


Figure 33-6: `PKTE_BUF_PTR` Register Diagram

Table 33-32: `PKTE_BUF_PTR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	OUTBUF	Output Buffer Pointer. The <code>PKTE_BUF_PTR.OUTBUF</code> bit field indicates the offset of the next address (entry) in the output buffer that will be written next by the packet engine. This bit field is reset to zero after starting up and decremented by 4 at every output buffer write operation. Pointers wrap around; the maximum value this field can have equals the output buffer size minus 4.
7:0 (R/NW)	INBUF	Input Buffer Pointer. The <code>PKTE_BUF_PTR.INBUF</code> bit field indicates the offset of the next address (entry) in the input buffer that will be read next by the packet engine. The bit field is reset to zero after starting up and incremented by 4 at every input buffer read operation. Pointers wrap around; the maximum value this field can have equals the input buffer size minus 4.

Packet Engine Buffer Threshold Register

When in autonomous ring mode or target command mode, the `PKTE_BUF_THRESH` register defines the high- and low-level value at which the packet engine starts to transfer packet data in or out of the internal packet buffers. These parameters can be used to control the DMA burst size for packet data input and output from the packet engine. In direct host mode, this register contains both threshold values to reduce the amount of packet engine interrupts.

The input buffer threshold (`ibufthrsh`) interrupt indicates that the input buffer counter is less than or equal to the input buffer threshold value set in this register - this interrupt can be used to wake up a process that stalled on a full input buffer.

The output buffer threshold (`obufthrsh`) interrupt indicates that the output buffer counter exceeds the output buffer threshold setting. The output buffer interrupt remains active until the output buffer counter is decremented to zero again.

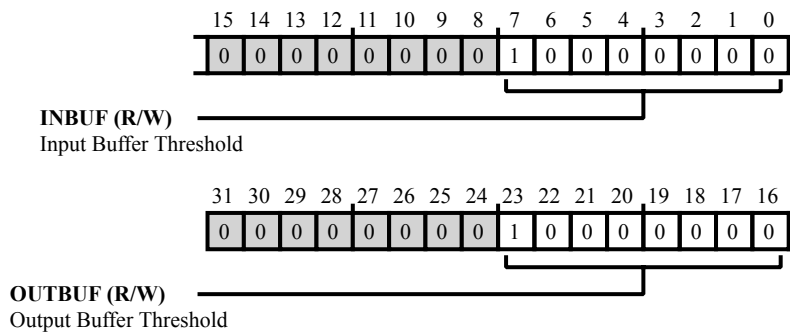


Figure 33-7: `PKTE_BUF_THRESH` Register Diagram

Table 33-33: `PKTE_BUF_THRESH` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	OUTBUF	<p>Output Buffer Threshold.</p> <p>The <code>PKTE_BUF_THRESH.OUTBUF</code> bit field specifies how many bytes must be available in the packet engine output buffer before an output transfer starts. Valid values range from 0 to 252, in multiples of 4.</p> <p>In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application.</p> <p>In direct host mode, the output buffer threshold (<code>obufthrsh</code>) interrupt activates when the output buffer counter for the output buffer exceeds the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.</p>

Table 33-33: PKTE_BUF_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	INBUF	<p>Input Buffer Threshold.</p> <p>The <code>PKTE_BUF_THRESH.INBUF</code> bit field specifies how many bytes must be free in the packet engine input buffer before an input transfer starts. Valid values range from 0 to 252, in multiples of 4.</p> <p>In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application.</p> <p>In direct host mode, the input buffer threshold (<code>ibufthrsh</code>) interrupt activates when the input buffer counter for the input buffer is below or equal the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.</p>

Packet Engine Command Descriptor Ring Base Address

The `PKTE_CDRBASE_ADDR` register holds the command descriptor ring base address in host memory. It is only applicable in autonomous ring mode. The `PKTE_CDRBASE_ADDR` register is ignored for all other modes when command descriptors are directly written into the descriptor registers.

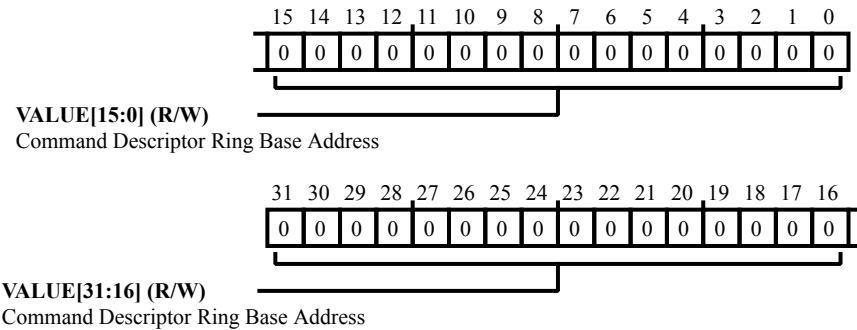


Figure 33-8: `PKTE_CDRBASE_ADDR` Register Diagram

Table 33-34: `PKTE_CDRBASE_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor Ring Base Address. The <code>PKTE_CDRBASE_ADDR.VALUE</code> bit field specifies the base location of the command descriptor ring in the host memory space.

Packet Engine Command Descriptor Count Register

The `PKTE_CDSC_CNT` register holds the counter for the number of descriptors in the Command Descriptor Ring (CDR). It is decremented by the packet engine each time a valid descriptor is read from the CDR and processed.

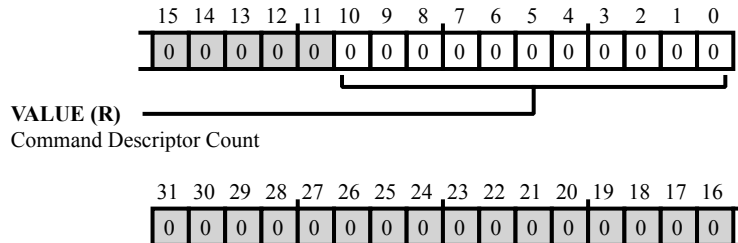


Figure 33-9: PKTE_CDSC_CNT Register Diagram

Table 33-35: PKTE_CDSC_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Command Descriptor Count. The <code>PKTE_CDSC_CNT.VALUE</code> bit field provides the number of command descriptors in the command descriptor ring. The packet engine decrements the counter when a valid command descriptor is read from the CDR and processed.

Packet Engine Command Descriptor Count Increment Register

The `PKTE_CDSC_INCR` register is accessible by the host connected through the system slave bus. The host can increment the command descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

In autonomous ring mode, the host must prepare 1 to 255 valid command descriptors in the CDR and then write this register with a value between 1 and 255. The write triggers the packet engine to fetch the command descriptors from the CDR. In direct host mode or target command mode, the host must write one valid command descriptor to the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is available.

A CDR threshold interrupt is activated when the command descriptor counter is less than or equal to the threshold value set in the `PKTE_RING_THRESH` register. This interrupt can be used to wake up a process that stalled on a full CDR.

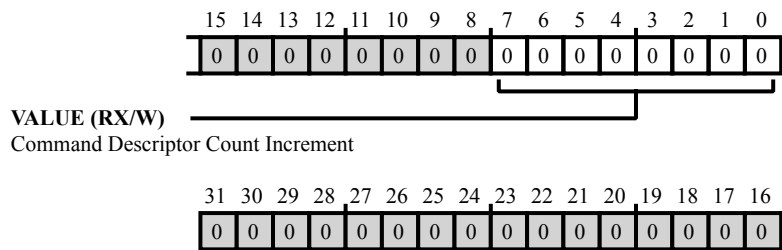


Figure 33-10: `PKTE_CDSC_INCR` Register Diagram

Table 33-36: `PKTE_CDSC_INCR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Command Descriptor Count Increment. The value written to the <code>PKTE_CDSC_INCR.VALUE</code> bit field is added to the command descriptor counter. The counter is protected against overflow (see the <code>PKTE_RING_STAT</code> register description). Note that bits[10:8] should be written with zeros.

Packet Engine Configuration Register

The `PKTE_CFG` register is used to select static settings that control the packet-processing path. This register is typically the last one to be written during the initialization sequence. These settings are typically set at initialization and not changed again.

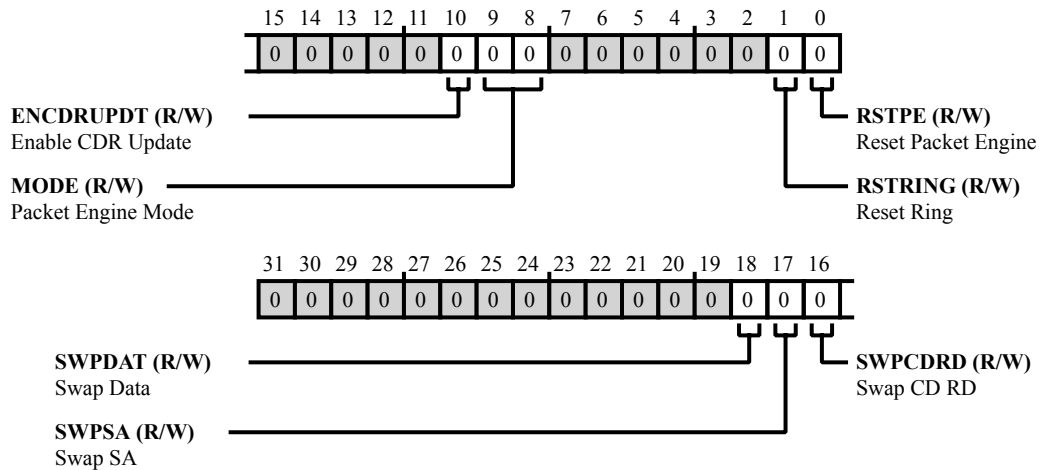


Figure 33-11: PKTE_CFG Register Diagram

Table 33-37: PKTE_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SWPDAT	Swap Data. The <code>PKTE_CFG.SWPDAT</code> bit enables endian swap for packet data as configured in the <code>PKTE_ENDIAN_CFG.MSTRBSWP</code> bits for the packet data DMA read and write.
		0 No Endian Swap
		1 Apply Endian Swap
17 (R/W)	SWPSA	Swap SA. The <code>PKTE_CFG.SWPSA</code> bit enables endian swap for a SA record as configured in the <code>PKTE_ENDIAN_CFG.MSTRBSWP</code> bits for the SA record and state record DMA read and write. If the <code>PKTE_ENDIAN_CFG.MSTRBSWP</code> bits specify no endian swap, this bit is ignored.
		0 No Endian Swap
		1 Apply Endian Swap

Table 33-37: PKTE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SWPCDRD	Swap CD RD. The PKTE_CFG.SWPCDRD bit enables endian swap for descriptors as configured in the PKTE_ENDIAN_CFG.MSTRBSWP bits for the command descriptor DMA read and result descriptor DMA write. If the PKTE_ENDIAN_CFG.MSTRBSWP bits specify no endian swap, this bit is ignored.
		0 No Endian Swap
		1 Apply Endian Swap
10 (R/W)	ENCDRUPDT	Enable CDR Update. The PKTE_CFG.ENCDRUPDT bit enables the packet engine to update, (clear the ownership bits) in the command descriptor in the CDR.
		0 Do Not Clear Ownership Bits. The packet engine does not clear the ownership bits in the command descriptor when it completes an operation. The host application must clear the ownership bits in "old descriptors" before the packet engine is allowed to wrap around the CDR to re-encounter these "old descriptors". This setting has the advantage of eliminating a separate DMA write to the CDR.
		1 Clear Ownership Bits. The packet engine clears (set to zero) the ownership bits in the current command descriptor in the CDR. This prevents the packet engine from re-processing an "old descriptor" when it wraps around the CDR.
9:8 (R/W)	MODE	Packet Engine Mode. The PKTE_CFG.MODE bit field selects how the packet engine receives commands.
		0 Direct Host Mode.
		1 Target Command Mode with Result Descriptor Ring Disabled.
		2 Target Command Mode with Result Descriptor Ring Enabled.
		3 Autonomous Ring Mode

Table 33-37: PKTE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	RSTRING	Reset Ring. The <code>PKTE_CFG.RSTRING</code> bit resets the internal counters for the CDR and RDR, <code>PKTE_CDSC_CNT</code> and <code>PKTE_RDSC_CNT</code> registers) to zero. Resets the <code>PKTE_RING_PTR</code> register to the base address. After the reset the rings are empty. This bit must be written with a '1' to reset the descriptor ring manager and then re-written with a '0' to release the reset. Note that this bit can remain in the reset state if the CDR ring is disabled (<code>PKTE_CFG.MODE</code> is not 0b11). Note that this reset must be coordinated with the 'owner' of the descriptor ring to ensure that the pointers are in sync after the reset.
		0 Release the Descriptor Ring Manager Reset
		1 Reset the Descriptor Ring Manager
0 (R/W)	RSTPE	Reset Packet Engine. The <code>PKTE_CFG.RSTPE</code> bit resets the packet engine and the state machine logic that drives header processing, DMA, and context management. The <code>PKTE_CFG.RSTPE</code> bit resets the <code>PKTE_CTL_STAT</code> and <code>PKTE_LEN</code> internal registers. This bit must be written with a 1 to reset the packet engine and then re-written with a 0 to release the reset. Note that this bit should not be used by a typical application. It is provided to use during development testing or to recover from critical errors. Note that the <code>PKTE_CTL_STAT.PADVAL</code> and <code>PKTE_CTL_STAT.PADCTLSTAT</code> bit fields are only reset when in autonomous ring mode, but the <code>PKTE_CTL_STAT.PRNGMD</code> bit is not reset. Halt mode is not affected by this reset as well. When exiting out of halt mode, a HW reset is required or a write to the <code>PKTE_CONT</code> register.
		0 Release the Packet Engine Reset
		1 Reset the Packet Engine

PE Clock Control Register

The `PKTE_CLK_CTL` register controls the clock enable signals. This register can be used to enable the clock for read and write access to SA registers or to enable the required clock signals for certain crypto functions. The setting of this register overrides the packet engine dynamic clock enable.

In autonomous ring mode and target command modes, this register can be all zeros; the packet engine dynamically requests the external clock manager to activate the module clocks. This register can be used in combination with the debugging interface for internal register access.

In direct host mode, the clock enable bits for the packet engine (`PKTE_CLK_CTL.ENPECLK`) and for ARC4 (`PKTE_CLK_CTL.ENARC4CLK`) must be enabled to write and read the SA record and state record registers. All module clocks that are required for the current operation must be enabled during processing.

Note that all the clocks are enabled by default to reset all the registers within the packet engine. After a system reset the host can program this register to disable clocks for power reduction.

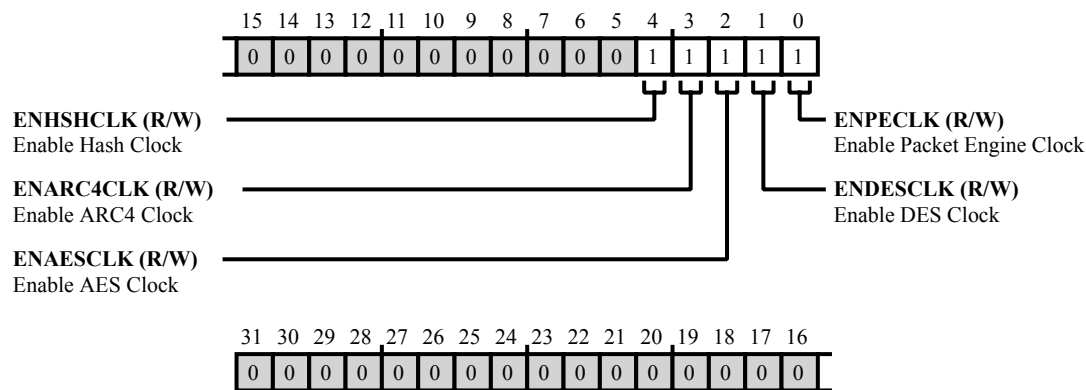


Figure 33-12: `PKTE_CLK_CTL` Register Diagram

Table 33-38: `PKTE_CLK_CTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	ENHSHCLK	Enable Hash Clock. The <code>PKTE_CLK_CTL.ENHSHCLK</code> bit enables the clock to the hash functions.
		0 Do not enable the hash clock
		1 Enable the hash clock
3 (R/W)	ENARC4CLK	Enable ARC4 Clock. The <code>PKTE_CLK_CTL.ENARC4CLK</code> bit enables the clock to the ARC4 function.
		0 Do not enable the ARC4 clock
		1 Enable the ARC4 clock

Table 33-38: PKTE_CLK_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	ENAESCLK	Enable AES Clock. The <code>PKTE_CLK_CTL.ENAESCLK</code> bit enables the clock to the AES encrypt/decrypt function.
		0 Do not enable the AES clock
		1 Enable the AES clock
1 (R/W)	ENDESCLK	Enable DES Clock. The <code>PKTE_CLK_CTL.ENDESCLK</code> bit enables the clock to the DES function.
		0 Do not enable the DES clock
		1 Enable the DES clock
0 (R/W)	ENPECLK	Enable Packet Engine Clock. The <code>PKTE_CLK_CTL.ENPECLK</code> bit enables the clock in the PKTE data path.
		0 Do not enable the PKTE data path clock
		1 Enable the PKTE data path clock

PKTE Continue Register

A write to the `PKTE_CONT` register (with any value) releases the packet engine from a halt state when in halt mode.

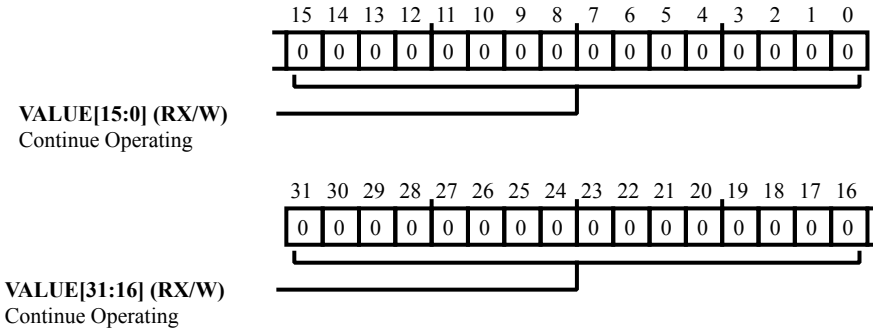


Figure 33-13: `PKTE_CONT` Register Diagram

Table 33-39: `PKTE_CONT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Continue Operating. The <code>PKTE_CONT.VALUE</code> bit field releases the packet engine from a halt state when written with any value in halt mode.

Packet Engine Control Register

The `PKTE_CTL_STAT` register has a dual function. Together with the data in the SA, this register provides the basic command information for the packet engine to process a packet. When the packet engine successfully or unsuccessfully completes an operation, the packet engine control/status register provides the result status for the host.

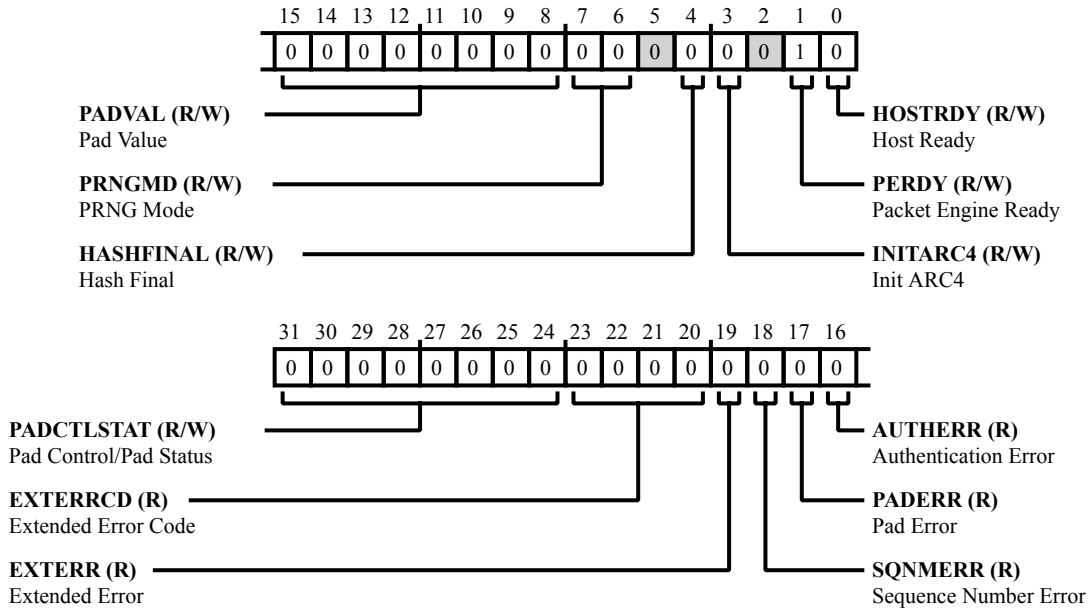


Figure 33-14: `PKTE_CTL_STAT` Register Diagram

Table 33-40: `PKTE_CTL_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	<code>PADCTLSTAT</code>	<p>Pad Control/Pad Status.</p> <p>The <code>PKTE_CTL_STAT.PADCTLSTAT</code> bit field is used to control the pad boundary for pad insertion (outbound) and after processing returns the number of inserted (outbound) or detected (inbound) pad bytes.</p> <p>For the command descriptor, the enumerations below provide the codes for the pad boundary for the outbound operations. This can be used for traffic flow security to conceal the number of payload bytes in an encrypted packet.</p> <p>For the result descriptor inbound operations that use pad types SSL, TLS, IPsec or PKCS#7, it returns the number of detected pad bytes. For all other inbound operations, it returns zero since the other pad modes do not allow implicit determination of pad count. If a pad verify failure occurs, it returns zero. For an outbound operation, it returns the number of inserted pad bytes for all pad types. The pad value includes added bytes such as the pad length and the next header field in an IPsec ESP pad type.</p>
		0 Align packet end to modulo 8-byte boundary
		1 Align packet end to modulo 1-byte boundary

Table 33-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Align packet end to modulo 4-byte boundary
		4	Align packet end to modulo 8-byte boundary
		8	Align packet end to modulo 16-byte boundary
		16	Align packet end to modulo 32-byte boundary
		32	Align packet end to modulo 64-byte boundary
		64	Align packet end to modulo 128-byte boundary
		128	Align packet end to modulo 256-byte boundary
23:20 (R/NW)	EXTERRCD	Extended Error Code. The PKTE_CTL_STAT.EXTERRCD bit field represents an encoded error condition.	
19 (R/NW)	EXTERR	Extended Error. The PKTE_CTL_STAT.EXTERR bit field provides an extended error code.	
		0	No Extended Error
		1	Extended Error
18 (R/NW)	SQNMERR	Sequence Number Error. The PKTE_CTL_STAT.SQNMERR bit indicates that for an inbound operation, there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition.	
		0	No Sequence Number Error
		1	Sequence Number Error
17 (R/NW)	PADERR	Pad Error. The PKTE_CTL_STAT.PADERR bit indicates that for an inbound operation the decrypted pad does not match the expected values.	
		0	No Pad Error
		1	Pad Error
16 (R/NW)	AUTHERR	Authentication Error. The PKTE_CTL_STAT.AUTHERR bit indicates that for an inbound operation the authentication value in the packet does not match the computed value.	
		0	No Authentication Error
		1	Authentication Error

Table 33-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration								
15:8 (R/W)	PADVAL	<p>Pad Value.</p> <p>The <code>PKTE_CTL_STAT.PADVAL</code> bit field is used to pass the pad value between the host and the packet engine.</p> <p>Command Descriptor: (write-only)</p> <p>For outbound operations that use pad type IPsec, the host must populate this field with the value that is to be inserted into the next header field. For the IPsec ESP operation, this next header is part of the ESP trailer of the innermost operation's header and the value must be 50 decimal. For outbound encrypt operations that use the pad type constant or constant SSL, the host must specify the fixed constant value in this field. For all other outbound and inbound operations, this field is not used.</p> <p>Result Descriptor: (read only)</p> <p>For inbound operations that use pad type IPsec, the packet engine returns the next header field that it detects. For IPsec ESP inbound operations, this is the next header field in the innermost operation's header, which will typically be the value for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other outbound operations, the packet engine will not update this field. For all other inbound operations, the returned pad value is zero.</p>								
7:6 (R/W)	PRNGMD	<p>PRNG Mode.</p> <p>The <code>PKTE_CTL_STAT.PRNGMD</code> bits select the pseudo-random number generator mode.</p> <table border="1"> <tbody> <tr> <td>0</td> <td>Operation does not use the PRNG function. Operation does not use the PRNG function.</td> </tr> <tr> <td>1</td> <td>PRNG Init. PRNG is initialized with a SEED, KEY and an LFSR value as defined in the SA.</td> </tr> <tr> <td>2</td> <td>PRNG Generate. Pseudo-random data is generated with the LFSR as input value. Before this mode can be used, the PRNG must be initialized with a valid SEED, KEY and LFSR using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).</td> </tr> <tr> <td>3</td> <td>PRNG Test. It can be used to test the PRNG function with custom input data. Before this mode can be used, the PRNG must be initialized once with a valid SEED using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).</td> </tr> </tbody> </table>	0	Operation does not use the PRNG function. Operation does not use the PRNG function.	1	PRNG Init. PRNG is initialized with a SEED, KEY and an LFSR value as defined in the SA.	2	PRNG Generate. Pseudo-random data is generated with the LFSR as input value. Before this mode can be used, the PRNG must be initialized with a valid SEED, KEY and LFSR using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).	3	PRNG Test. It can be used to test the PRNG function with custom input data. Before this mode can be used, the PRNG must be initialized once with a valid SEED using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).
0	Operation does not use the PRNG function. Operation does not use the PRNG function.									
1	PRNG Init. PRNG is initialized with a SEED, KEY and an LFSR value as defined in the SA.									
2	PRNG Generate. Pseudo-random data is generated with the LFSR as input value. Before this mode can be used, the PRNG must be initialized with a valid SEED, KEY and LFSR using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).									
3	PRNG Test. It can be used to test the PRNG function with custom input data. Before this mode can be used, the PRNG must be initialized once with a valid SEED using PRNG Init (<code>PKTE_CTL_STAT.PRNGMD=b'01</code>).									

Table 33-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	HASHFINAL	Hash Final. When the <code>PKTE_CTL_STAT.HASHFINAL</code> bit is zero, the data to be hashed must be a multiple of the hash block size, 64 bytes for SHA-1, MD5, SHA-224, SHA-256. This bit is only applicable for Basic Hash, Basic Encrypt-Hash and Basic Hash-De-encrypt operations that use the SHA-1, MD5, SHA-224, SHA-256 hash algorithm. The <code>PKTE_CTL_STAT.HASHFINAL</code> bit is overruled for HMAC operations that always completes the hash and always returns the last written value on a read by the host.
		0 Perform Intermediate Hash Operation. The packet engine performs an intermediate hash operation by generating an intermediate hash digest on the data presented on the input. No hash pad is applied.
		1 Perform Final Hash Operation. The packet engine appends the required final hash pad and generates the final hash digest on the data presented on the input. This completes the hash operation.
3 (R/W)	INITARC4	Init ARC4. The <code>PKTE_CTL_STAT.INITARC4</code> bit initializes the ARC4 crypto algorithm with a new key. This bit always returns the last written value on a read by the Host. This bit is only applicable for operations that use the ARC4 algorithm and must be zero for all other operations.
		0 Load ARC4 State and ARC4 i/j pointer from the SA The ARC4 State and ARC4 i/j pointer are loaded from the SA to continue the encrypt/decrypt processing from the previous algorithm state.
		1 Read ARC4 key from the SA-record and initialize ARC4 S-boxes using this key The ARC4 key is read from the SA-record, the ARC4 S-boxes are initialized using this key, prior to the encryption/decryption of data. This bit overrules Stateful mode as defined in bits [9:8] of <code>PKTE_SA_CMD1</code> .
1 (R/W)	PERDY	Packet Engine Ready. The <code>PKTE_CTL_STAT.PERDY</code> bit indicates that the packet engine has completed processing the command descriptor and returns the result descriptor with ownership set to the host. This bit can be reset to 0 by the host and the packet engine, but only the packet engine can set this bit. When the packet engine is idle (not processing), this bit always returns '1' on a read by the host.

Table 33-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	HOSTRDY	<p>Host Ready.</p> <p>The <code>PKTE_CTL_STAT.HOSTRDY</code> bit indicates that the host has populated the command descriptor. This bit can be reset to 0 by the host and the packet engine, but only the host can set this bit. When the packet engine is idle (not processing), this bit always returns '0' on a read by the host.</p>

Starting Entry of 256-byte Data Input/Output Buffer

When in direct host mode, the source packet data is written here to be transferred to the packet engine. The host can monitor the available space in the input buffer through the `PKTE_STAT` register. This is also the location in the packet engine from where output data is read when in direct host mode. The host can monitor the available bytes in the output buffer through the `PKTE_STAT` register.

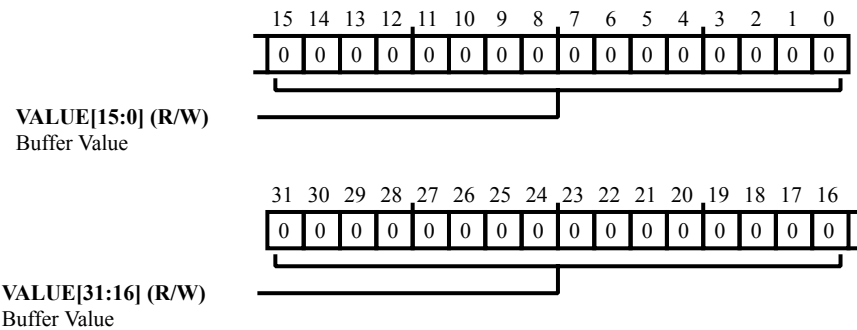


Figure 33-15: `PKTE_DATAIO_BUF` Register Diagram

Table 33-41: `PKTE_DATAIO_BUF` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Buffer Value.

Packet Engine Destination Address

The `PKTE_DEST_ADDR` register holds the starting (byte) address to write the result packet from the requested operation.

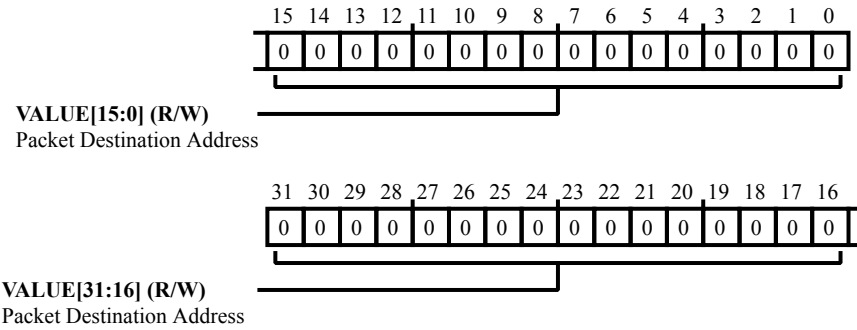


Figure 33-16: `PKTE_DEST_ADDR` Register Diagram

Table 33-42: `PKTE_DEST_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Destination Address.

Packet Engine DMA Configuration Register

The `PKTE_DMA_CFG` register configures the maximum burst transfer size, enables incremental transfers, and inserts IDLE cycles between two bus transfers.

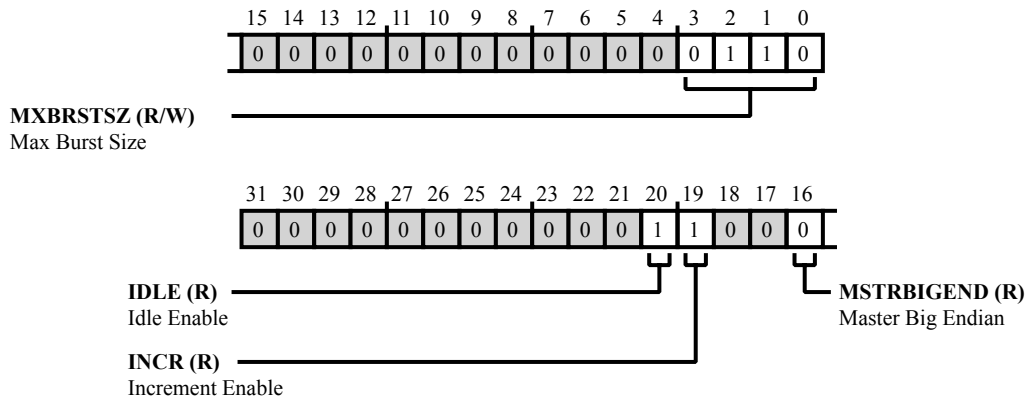


Figure 33-17: PKTE_DMA_CFG Register Diagram

Table 33-43: PKTE_DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	IDLE	Idle Enable. The <code>PKTE_DMA_CFG.IDLE</code> bit allows the peripheral bus master to insert one additional IDLE transfer between two successive peripheral bus master burst operations. This provides the arbiter one additional cycle to hand over the grant to another peripheral bus master.
		0 The peripheral bus master inserts no IDLE cycle between two successive burst operations
		1 The peripheral bus master inserts one additional IDLE transfer between two successive burst operations

Table 33-43: PKTE_DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/NW)	INCR	<p>Increment Enable.</p> <p>The <code>PKTE_DMA_CFG.INCR</code> bit lets the peripheral bus master generate INC4, INC8 and INC16 type of burst transfers.</p> <p>By default, the peripheral bus master generates the largest possible incremental burst of unspecified length (INCR) with a maximum length (in bytes) as configured by the <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field. In case there are less than 4 bytes of data available or the 1kB boundary will be crossed using a burst operation, then a single transfer of size byte is generated.</p> <p>When the <code>PKTE_DMA_CFG.INCR</code> bit is set, the peripheral bus master generates one or more incremental burst of specified length (INC4, INC8, INC16). In case there is less data available then the smallest possible burst (INC4) or the 1kB boundary will be crossed using a burst operation, then an unspecified length burst or a single transfer of size byte is generated.</p>
		0 The bus master will generate only INCR burst types
		1 The bus master will generate INC4, INC8 and INC16 burst types
16 (R/NW)	MSTRBIGEND	<p>Master Big Endian.</p> <p>The <code>PKTE_DMA_CFG.MSTRBIGEND</code> bit determines whether the engine is used in a little or big endian system.</p>
		0 Little endian
		1 Big endian
3:0 (R/W)	MXBRSTSZ	<p>Max Burst Size.</p> <p>The <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field configures the maximum size of an unspecified length burst (INC) at the bus in bytes. When there is less data available than the <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field setting or the 1kB boundary will be crossed using a burst operation, then the length of the burst can be less than <code>PKTE_DMA_CFG.MXBRSTSZ</code>. Any requested transfers larger than this size are broken up in to multiple burst transfers of this size or less.</p>

Packet Engine Endian Configuration Register

The packet engine incorporates a powerful interface specific endian handler. This endian handler allows byte lane swapping in each direction for data passing through the host interface.

The `PKTE_ENDIAN_CFG` register configures the byte order function for the peripheral bus master and peripheral bus slave interface. The bits for the peripheral bus master are combined in four sets of two bits; each group configures a byte swap function for a particular DMA transfer. The same applies for the peripheral bus slave interface.

The `PKTE_ENDIAN_CFG` register also defines the endian swapping that occurs for host-initiated target transfers and for packet engine master DMA read and write transfers. Individual endian swap enable bits in the configuration (`PKTE_CFG`) register can enable the endian swap for various transaction types: command descriptors and result descriptors, SA records and state records, packet data.

In direct host mode, only target operations are supported. Only the target endian configuration of this register is applicable.

Note: This register is typically programmed once during the initialization phase, although software is allowed to dynamically change the setting in this register just before initiating a data transfer. The developer will have to analyze the benefit of the cycles needed to write the endian register dynamically versus handling endian swapping for some data structures in the host system (most modern processors support a byte swap in zero cycles). Certainly the endian swap should be set correctly for the packet data, since this represents the majority of the data transferred.

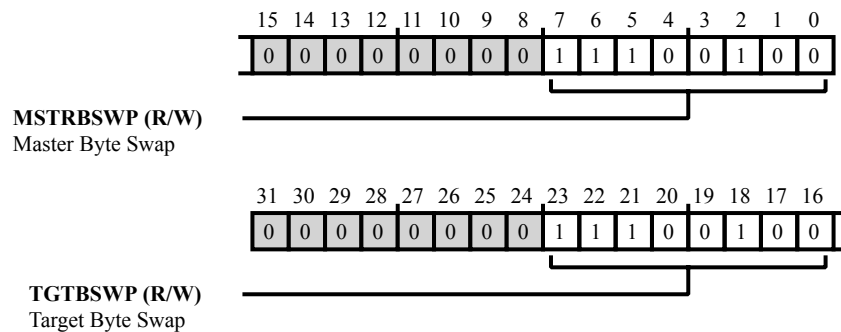


Figure 33-18: `PKTE_ENDIAN_CFG` Register Diagram

Table 33-44: `PKTE_ENDIAN_CFG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TGTBSWP	<p>Target Byte Swap.</p> <p>The <code>PKTE_ENDIAN_CFG.TGTBSWP</code> bit field configures the byte swap for peripheral bus target transfers. Note that only target word transfers are supported. Each double-bit field in this register specifies the source of the indicated byte lane. The field values are interpreted as follows: 00 = byte 0, 01 = byte 1, 10 = byte 2, 11 = byte 3.</p> <p>Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).</p>

Table 33-44: PKTE_ENDIAN_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MSTRBSWP	<p>Master Byte Swap.</p> <p>The <code>PKTE_ENDIAN_CFG.MSTRBSWP</code> bit field configures the byte swap for peripheral bus master multi-byte transfers, including command descriptors, result descriptors, SA records, state records and packet data. Separate controls in the PKTE_CFG register can enable this swap individually for each of the 4 types of data. Each double-bit field in this register specifies the source of the indicated peripheral bus byte lane. The field values are interpreted as follows: 00=byte 0, 01=byte 1, 10=byte 2, 11=byte 3.</p> <p>Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).</p>

Packet Engine Halt Control Register

The `PKTE_HLT_CTL` register controls the packet engine halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. During the halt mode, the host can read all internal registers for examination without side-effects. When halted, the host should not write to any registers. To continue packet engine operation, the host must write to the `PKTE_CONT` (PKTE continue) register.

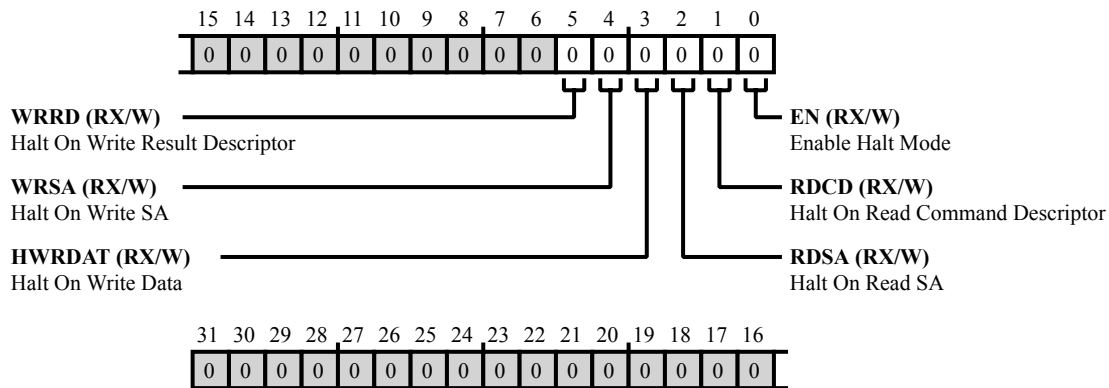


Figure 33-19: `PKTE_HLT_CTL` Register Diagram

Table 33-45: `PKTE_HLT_CTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W)	WRRD	Halt On Write Result Descriptor. The <code>PKTE_HLT_CTL.WRRD</code> bit halts the packet engine in the <code>HALT_WRITE_STATUS</code> state after it completes a result descriptor write operation to the result descriptor ring. The host can use this bit to examine the result descriptor that is currently in the host memory.
		0 Do not halt the Packet Engine operation
		1 Halt the Packet Engine operation
4 (RX/W)	WRSA	Halt On Write SA. The <code>PKTE_HLT_CTL.WRSA</code> bit halts the packet engine in the <code>HALT_WRITE_SA</code> state after it completes an SA write operation to the host memory. The host can use this bit to examine the security context that is currently in the host memory.
		0 Do not halt the Packet Engine operation
		1 Halt the Packet Engine operation

Table 33-45: PKTE_HLT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (RX/W)	HWRDAT	Halt On Write Data. The <code>PKTE_HLT_CTL.HWRDAT</code> bit halts the packet engine in the <code>HALT_DATA</code> state after it completes writing the result packet data to the host memory. The host can use this bit to examine the result packet that is currently in the host memory.
		0 Do not halt the Packet Engine operation
		1 Halt the Packet Engine operation
2 (RX/W)	RDSA	Halt On Read SA. The <code>PKTE_HLT_CTL.RDSA</code> bit halts the packet engine in the <code>HALT_READ_SA</code> state after it completes an SA read operation from the host memory. The host can use this bit to examine the security context that is currently in the SA registers.
		0 Do not halt the Packet Engine operation
		1 Halt the Packet Engine operation
1 (RX/W)	RD CD	Halt On Read Command Descriptor. The <code>PKTE_HLT_CTL.RD CD</code> bit halts the packet engine in the <code>HALT_READ_DESCR</code> state after it completes a command descriptor read operation from the command descriptor ring. It will halt whether the descriptor is valid or invalid. The host can use this bit to examine the command descriptor that is currently in the internal command descriptor registers.
		0 Do not halt the Packet Engine operation
		1 Halt the Packet Engine operation
0 (RX/W)	EN	Enable Halt Mode. The <code>PKTE_HLT_CTL.EN</code> bit enables halt mode where the packet engine can halt processing at any processing state as indicated by bits [5:1]. When halted, the packet engine continues operation on a write to the <code>PKTE_CONT</code> register.
		0 Do not enable halt mode
		1 Enable halt mode

Packet Engine Halt Status Register

The `PKTE_HLT_STAT` register reflects the status of the packet engine in halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. When the packet engine is halted, the host can read all internal registers for examination without side effects. The host should not write to any registers.

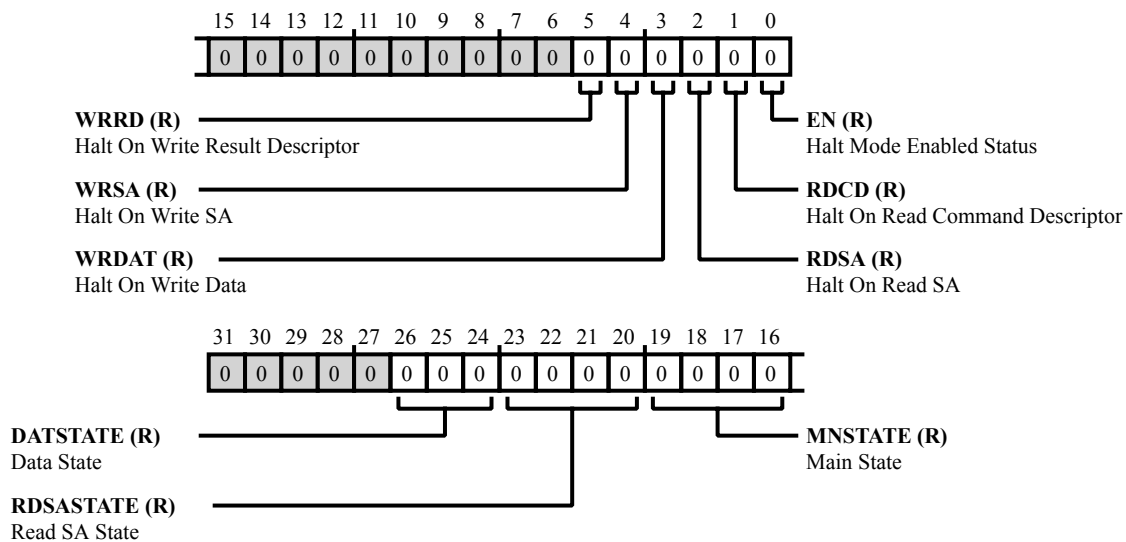


Figure 33-20: `PKTE_HLT_STAT` Register Diagram

Table 33-46: `PKTE_HLT_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26:24 (R/NW)	DATSTATE	Data State. The <code>PKTE_HLT_STAT.DATSTATE</code> bit field indicates the state of the packet engine read data FSM.
		0 DATA_IDLE, no operation
		1 DATA_READ
		2 DATA_WRITE
		3 DATA_WAIT
		5 DATA_PAD_READ
		6 DATA_BYP_READ
		7 RESERVED
23:20 (R/NW)	RDSASTATE	Read SA State. The <code>PKTE_HLT_STAT.RDSASTATE</code> bit field indicates the state of the packet engine read SA FSM.

Table 33-46: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		0 SA_IDLE, no operation
		1 SA_READ_CMD
		2 SA_READ_STATE_IV
		3-5 RESERVED
		6 SA_READ_ARC4_STATE
		7 SA_READ_WAIT
		8 RESERVED
		9 SA_WRITE_PROT_HDR
		10 RESERVED
		11 SA_WRITE_IV
		12 SA_WRITE_DIGEST
		13 SA_WRITE_ARC4_IJ_PNTR
		14 SA_WRITE_ARC4_STATE
		15 SA_WRITE_WAIT
19:16 (R/NW)	MNSTATE	<p>Main State.</p> <p>The <code>PKTE_HLT_STAT.MNSTATE</code> bit field indicates the state of the packet engine main FSM.</p>
		0 MAIN_IDLE, no operation
		1 MAIN_READ_CD, reading command descriptor
		2 MAIN_READ_SA, reading SA
		3 MAIN_DATA, processing data
		4 MAIN_WRITE_SA, writing SA
		5 MAIN_WRITE_STATUS, writing status
		6 MAIN_WRITE_CD, updating command descriptor
		7 MAIN_WRITE_RD, updating result descriptor
		8 MAIN_INIT_WAIT, wait single clock
		9 MAIN_HALT_READ_CD, halt after read command descriptor
		10 MAIN_HALT_READ_SA, halt after read SA
		11 MAIN_HALT_DATA, halt after processing data
		12 MAIN_HALT_WRITE_SA, halt after write SA

Table 33-46: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		13	MAIN_WAIT_FOR_CLOCK, wait for clocks to be active
		15	MAIN_HALT_WRITE_RD, halt after write result descriptor
5 (R/NW)	WRRD	Halt On Write Result Descriptor. The PKTE_HLT_STAT.WRRD bit reflects the value in the PKTE_HLT_CTL.WRRD bit.	
4 (R/NW)	WRSA	Halt On Write SA. The PKTE_HLT_STAT.WRSA bit reflects the value in the PKTE_HLT_CTL.WRSA bit.	
3 (R/NW)	WRDAT	Halt On Write Data. The PKTE_HLT_STAT.WRDAT bit reflects the value in the PKTE_HLT_CTL.HWRDAT bit.	
2 (R/NW)	RDSA	Halt On Read SA. The PKTE_HLT_STAT.RDSA bit reflects the value in the PKTE_HLT_CTL.RDSA bit.	
1 (R/NW)	RDCD	Halt On Read Command Descriptor. The PKTE_HLT_STAT.RDCD bit reflects the value in the PKTE_HLT_CTL.RDCD bit.	
0 (R/NW)	EN	Halt Mode Enabled Status.	
		0	Halt mode not enabled
		1	Halt mode enabled

Interrupt Mask Disable Register

The host can use the `PKTE_IMSK_DIS` register to clear individual bits in the `PKTE_INT_EN` register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 clears the interrupt enable bit, a 0 does not affect the interrupt enable bit in the `PKTE_INT_EN` register. Clearing the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

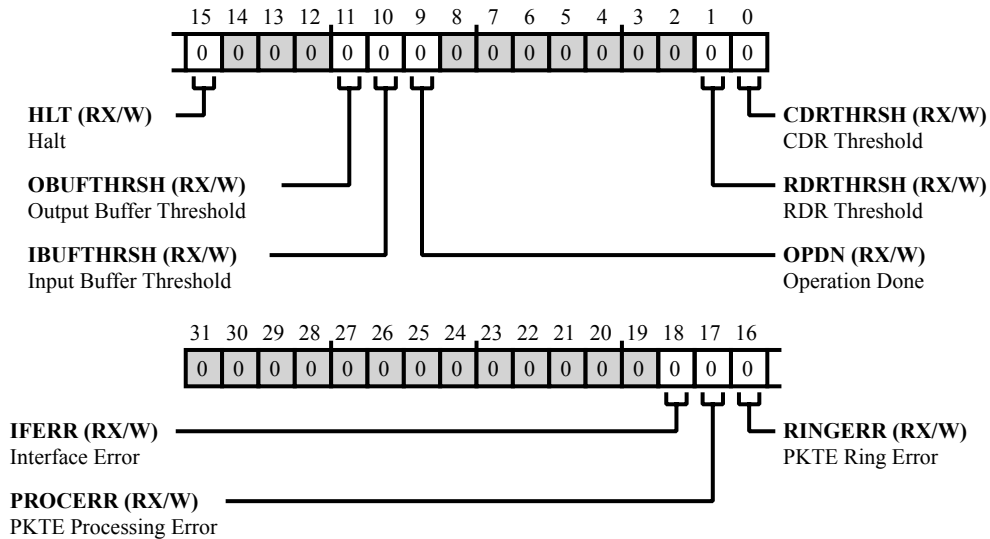


Figure 33-21: PKTE_IMSK_DIS Register Diagram

Table 33-47: PKTE_IMSK_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Write the <code>PKTE_IMSK_DIS</code> . <code>IFERR</code> bit to clear when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Write the <code>PKTE_IMSK_DIS</code> . <code>PROCERR</code> bit to clear an extended error that occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. Write the <code>PKTE_IMSK_DIS</code> . <code>RINGERR</code> bit to clear a CDR overflow or an RDR underflow.
15 (RX/W)	HLT	Halt. Write the <code>PKTE_IMSK_DIS</code> . <code>HLT</code> bit to clear when the packet engine is in the halt state.

Table 33-47: PKTE_IMSK_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. Write the <code>PKTE_IMSK_DIS.OBUFTHRSH</code> bit to clear the output buffer counter exceeds the output buffer threshold value defined in <code>PKTE_BUF_THRESH.OUTBUF</code> bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Write the <code>PKTE_IMSK_DIS.IBUFTHRSH</code> bit to clear when the input buffer counter is less than or equal to the input buffer threshold value defined in <code>PKTE_BUF_THRESH.INBUF</code> bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Write the <code>PKTE_IMSK_DIS.RDRTHRSH</code> bit to clear when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the <code>PKTE_RING_THRESH.RDRTHRSH</code> bit, or the RD counter for the RDR in the <code>PKTE_RDSC_CNT</code> register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Write the <code>PKTE_IMSK_DIS.CDRTHRSH</code> bit to clear when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the <code>PKTE_RING_THRESH.CDRTHRSH</code> bit.

Interrupt Mask Enable Register

The host can use the `PKTE_IMSK_EN` register to set individual bits in the `PKTE_INT_EN` register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 sets the interrupt enable bit, a 0 does not affect the interrupt enable bit in the `PKTE_INT_EN` register. Setting the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

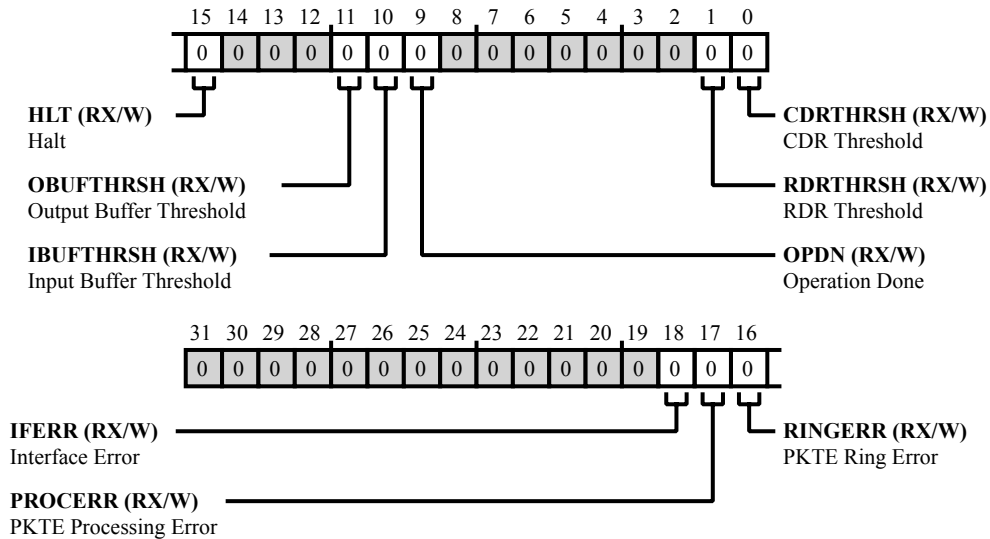


Figure 33-22: `PKTE_IMSK_EN` Register Diagram

Table 33-48: `PKTE_IMSK_EN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Set the <code>PKTE_IMSK_EN</code> . <code>IFERR</code> bit to indicate a host request for a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Set the <code>PKTE_IMSK_EN</code> . <code>PROCERR</code> bit to indicate an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error.
15 (RX/W)	HLT	Halt. Set the <code>PKTE_IMSK_EN</code> . <code>HLT</code> bit to indicate when the packet engine is in the halt state.

Table 33-48: PKTE_IMSK_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. Set the PKTE_IMSK_EN.OBUFTHRSH bit to indicate that the output buffer counter exceeds the output buffer threshold value defined in the PKTE_BUF_THRESH.OUTBUF bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Set the PKTE_IMSK_EN.IBUFTHRSH bit to indicate the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Set the PKTE_IMSK_EN.RDRTHRSH bit to indicate when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Set the PKTE_IMSK_EN.CDRTHRSH bit to indicate when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Masked Status Register

The `PKTE_IMSK_STAT` register provides interrupt status visibility to the host, after the interrupt mask is applied. This lets the host view the selected sources of interrupts that are directed to the interrupt output signal, that is connected to the system interrupt controller. As with the unmasked status register, all interrupt bits are latched and must be cleared using the `PKTE_INT_CLR` register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

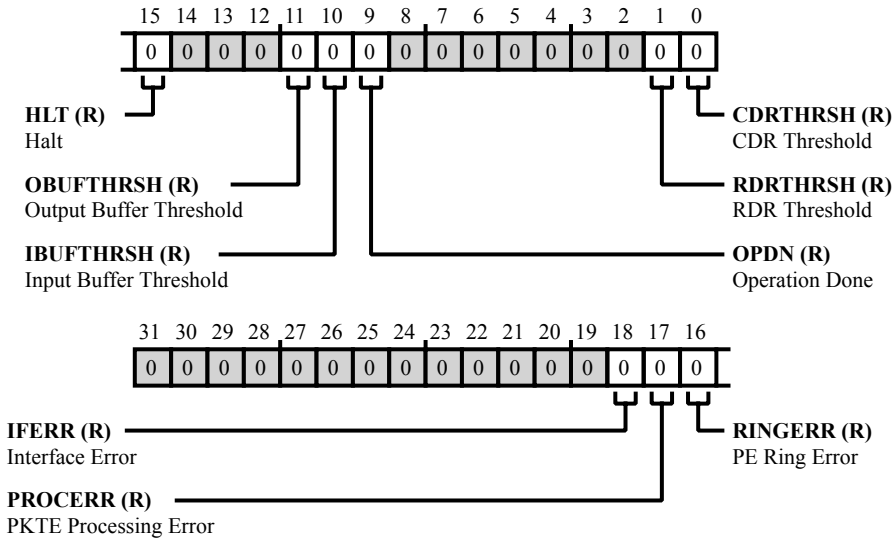


Figure 33-23: PKTE_IMSK_STAT Register Diagram

Table 33-49: PKTE_IMSK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The <code>PKTE_IMSK_STAT</code> .IFERR bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The <code>PKTE_IMSK_STAT</code> .PROCERR bit is when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PE Ring Error. The <code>PKTE_IMSK_STAT</code> .RINGERR bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The <code>PKTE_IMSK_STAT</code> .HLT bit is set when the packet engine is in the HALT state.

Table 33-49: PKTE_IMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	OBUFTHRSH	Output Buffer Threshold. The <code>PKTE_IMSK_STAT.OBUFTHRSH</code> bit is set when the output buffer counter exceeds the output buffer threshold value defined in <code>PKTE_BUF_THRESH.OUTPUTBUF</code> bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The <code>PKTE_IMSK_STAT.IBUFTHRSH</code> bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in <code>PKTE_BUF_THRESH.INBUF</code> bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The <code>PKTE_IMSK_STAT.RDRTHRSH</code> bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the <code>PKTE_RING_THRESH.RDRTHRSH</code> bit, or the RD counter for the RDR in <code>PKTE_RDSC_CNT</code> register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The <code>PKTE_IMSK_STAT.CDRTHRSH</code> bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the <code>PKTE_RING_THRESH.CDRTHRSH</code> bit.

Packet Engine Input Buffer Count Register

The `PKTE_INBUF_CNT` register provides the number of bytes available in the input buffer. The `PKTE_INBUF_CNT` register is used in direct host mode only.

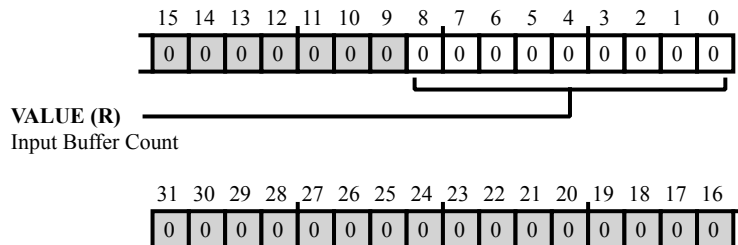


Figure 33-24: `PKTE_INBUF_CNT` Register Diagram

Table 33-50: `PKTE_INBUF_CNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Input Buffer Count. The <code>PKTE_INBUF_CNT.VALUE</code> bit field provides the number of bytes in the input buffer. The packet engine decrements the counter by 4 when a 32-bit word is read from the input buffer.

Packet Engine Input Buffer Count Increment Register

A host connected through the system slave bus can increment the input buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The `PKTE_INBUF_INCR` register is used in direct host mode only.

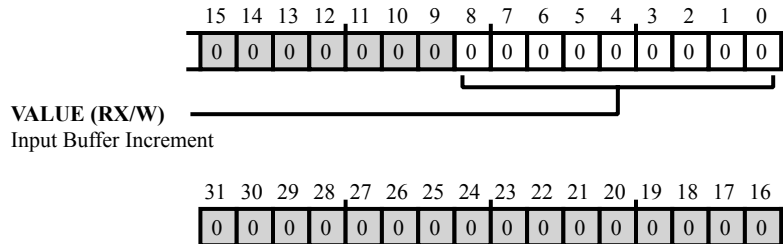


Figure 33-25: PKTE_INBUF_INCR Register Diagram

Table 33-51: PKTE_INBUF_INCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Input Buffer Increment. The value written is added to the input buffer counter. Valid values range from 4 to 256, in multiples of 4.

Interrupt Configuration Register

The `PKTE_INT_CFG` register configures the interrupt type that is sent to the interrupt line connected to the system interrupt controller. (Note that this only effects the final output of the interrupt subsystem).

Configuring the interrupt output type for pulse causes the interrupt signal to pulse low for two clock cycles when activated. When set for level, the interrupt signal is set low until cleared by the host (it follows the bit in the masked status register). For the host, this is typically set to level.

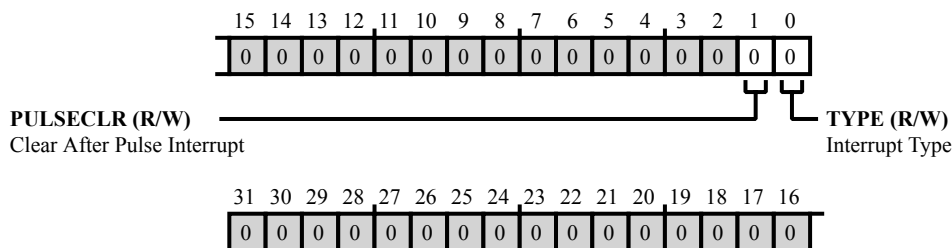


Figure 33-26: `PKTE_INT_CFG` Register Diagram

Table 33-52: `PKTE_INT_CFG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	PULSECLR	Clear After Pulse Interrupt. The <code>PKTE_INT_CFG.PULSECLR</code> bit clears the latched interrupt source after the pulse interrupt.
		0 Manually clear pulse interrupt source. Do not automatically clear the interrupt sources after pulsing the interrupt output. Clear the source by writing to the <code>PKTE_INT_CLR</code> register.
		1 Automatically clear pulse interrupt source. After pulsing the interrupt output, automatically clear the sources.
0 (R/W)	TYPE	Interrupt Type. The <code>PKTE_INT_CFG.TYPE</code> bit selects the type, pulse or level, for the interrupt output to the system.
		0 Level. The interrupt output is a level signal that is set low when an enabled interrupt is active until the interrupt is cleared.
		1 Pulse. The interrupt output is a two clock cycle low-active pulse, activated when an enabled interrupt is active.

Interrupt Clear Register

The `PKTE_INT_CLR` register allows the host processor to clear pending interrupts. A 1 written to a given bit in this register clears the corresponding interrupt. A 0 leaves the interrupt latch unchanged for that position.

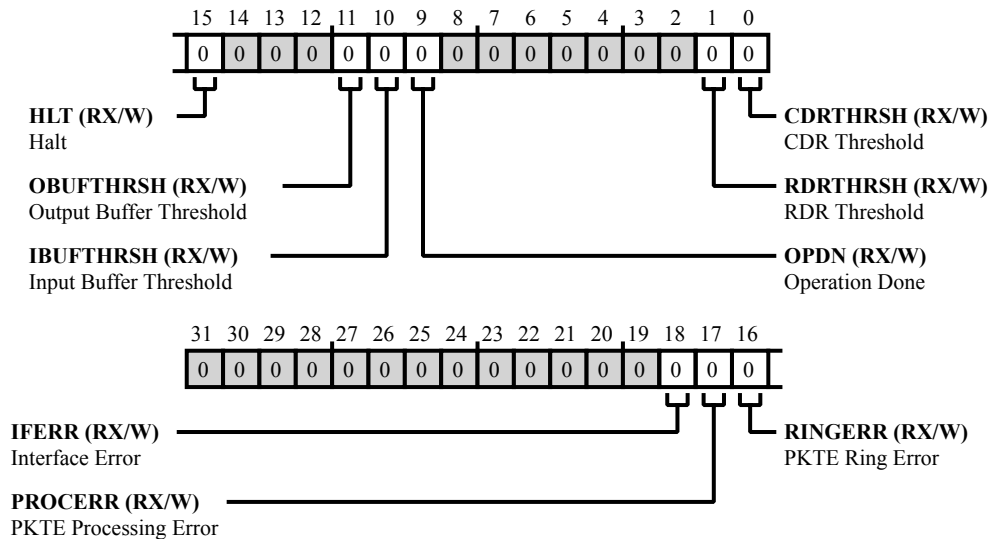


Figure 33-27: `PKTE_INT_CLR` Register Diagram

Table 33-53: `PKTE_INT_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. The <code>PKTE_INT_CLR</code> . <code>IFERR</code> bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. The <code>PKTE_INT_CLR</code> . <code>PROCERR</code> bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. The <code>PKTE_INT_CLR</code> . <code>RINGERR</code> bit is set on a CDR overflow or an RDR under-flow.
15 (RX/W)	HLT	Halt. The <code>PKTE_INT_CLR</code> . <code>HLT</code> bit is set when the packet engine is in the HALT state.
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. The <code>PKTE_INT_CLR</code> . <code>OBUFTHRSH</code> bit is set when the output buffer counter exceeds the output buffer threshold value defined in <code>PKTE_BUF_THRESH</code> . <code>OUTBUF</code> bit.

Table 33-53: PKTE_INT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. The <code>PKTE_INT_CLR.IBUFTHRSH</code> bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in <code>PKTE_BUF_THRESH.INBUF</code> bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. The <code>PKTE_INT_CLR.RDRTHRSH</code> bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the <code>PKTE_RING_THRESH.RDRTHRSH</code> bit, or the RD counter for the RDR in <code>PKTE_RDSC_CNT</code> register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. The <code>PKTE_INT_CLR.CDRTHRSH</code> bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the <code>PKTE_RING_THRESH.CDRTHRSH</code> bit.

Interrupt Enable Register

The `PKTE_INT_EN` register configures the interrupt mask for the host interrupt. This register is a bitmap for each of the possible interrupt sources. A 1 enables the interrupt source and a 0 disables the source. If an interrupt source is disabled, a cleared bit also clears the matching interrupt in the `PKTE_IMSK_STAT` register.

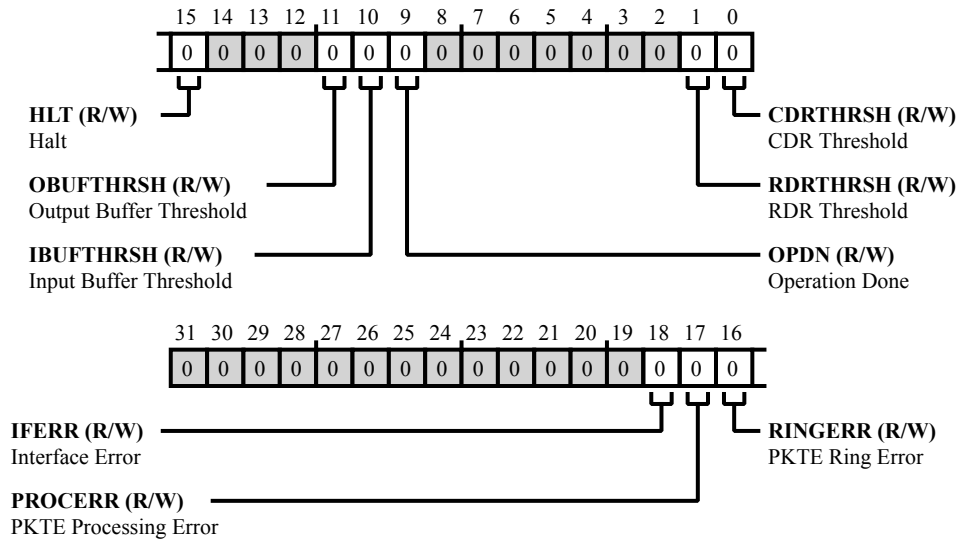


Figure 33-28: `PKTE_INT_EN` Register Diagram

Table 33-54: `PKTE_INT_EN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	IFERR	Interface Error. Set the <code>PKTE_INT_EN</code> . <code>IFERR</code> bit for host requests for a non 32-bit access to the packet engine interrupt or when the packet engine receives an error writing data back out to the host memory system.
17 (R/W)	PROCERR	PKTE Processing Error. Set the <code>PKTE_INT_EN</code> . <code>PROCERR</code> bit to enable the extended error occurred before, during or after processing the current packet in the packet engine interrupt.
16 (R/W)	RINGERR	PKTE Ring Error. Set the <code>PKTE_INT_EN</code> . <code>RINGERR</code> bit to enable the CDR overflow or RDR under-flow interrupt.
15 (R/W)	HLT	Halt. Set the <code>PKTE_INT_EN</code> . <code>HLT</code> bit for when the packet engine is in the HALT state.
11 (R/W)	OBUFTHRSH	Output Buffer Threshold. Set the <code>PKTE_INT_EN</code> . <code>OBUFTHRSH</code> bit for to trigger an interrupt when the output buffer counter exceeds the output buffer threshold value defined in the <code>PKTE_BUF_THRESH</code> . <code>OUTBUF</code> bit.

Table 33-54: PKTE_INT_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	IBUFTHRSH	Input Buffer Threshold. Set the PKTE_INT_EN.IBUFTHRSH bit for to trigger an interrupt when the input buffer counter is less than or equal to the input buffer threshold value defined in the PKTE_BUF_THRESH.INBUF bit.
9 (R/W)	OPDN	Operation Done.
1 (R/W)	RDRTHRSH	RDR Threshold. Set the PKTE_INT_EN.RDRTHRSH bit for to trigger an interrupt when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/W)	CDRTHRSH	CDR Threshold. Set the PKTE_INT_EN.CDRTHRSH bit for to trigger an interrupt when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Unmasked Status Register

The `PKTE_IUMSK_STAT` register provides interrupt status visibility to the host, prior to the interrupt mask being applied. Using this register, the host can view all potential sources of incoming interrupts. All of these sources, whether masked in or out, are latched in this register and must be cleared using the `PKTE_INT_CLR` register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

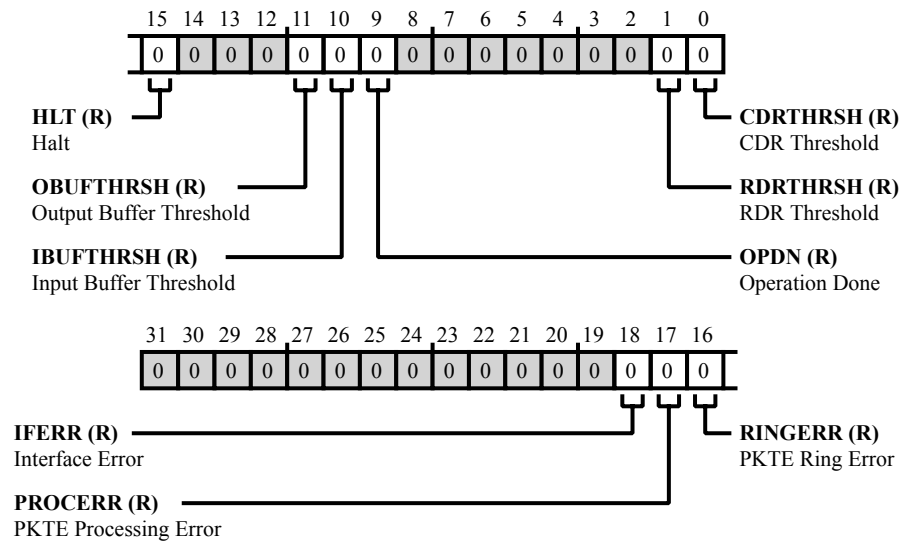


Figure 33-29: `PKTE_IUMSK_STAT` Register Diagram

Table 33-55: `PKTE_IUMSK_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The <code>PKTE_IUMSK_STAT</code> . <code>IFERR</code> bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The <code>PKTE_IUMSK_STAT</code> . <code>PROCERR</code> bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PKTE Ring Error. The <code>PKTE_IUMSK_STAT</code> . <code>RINGERR</code> bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The <code>PKTE_IUMSK_STAT</code> . <code>HLT</code> bit is set when the packet engine is in the HALT state.

Table 33-55: PKTE_IUMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	OBUFTHRSH	Output Buffer Threshold. The <code>PKTE_IUMSK_STAT.OBUFTHRSH</code> interrupt is triggered when the output buffer counter exceeds the output buffer threshold value defined in <code>PKTE_BUF_THRESH.OUTBUF</code> bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The <code>PKTE_IUMSK_STAT.IBUFTHRSH</code> interrupt is triggered when the input buffer counter is less than or equal to the input buffer threshold value defined in <code>PKTE_BUF_THRESH.INBUF</code> bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The <code>PKTE_IUMSK_STAT.RDRTHRSH</code> bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the <code>PKTE_RING_THRESH.RDRTHRSH</code> , or the RD counter for the RDR in <code>PKTE_RDSC_CNT</code> register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The <code>PKTE_IUMSK_STAT.CDRTHRSH</code> bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the <code>PKTE_RING_THRESH.CDRTHRSH</code> bit.

Packet Engine Length Register

The `PKTE_LEN` register gives the length of the packet, the bypass data and a second set of ownership bits.

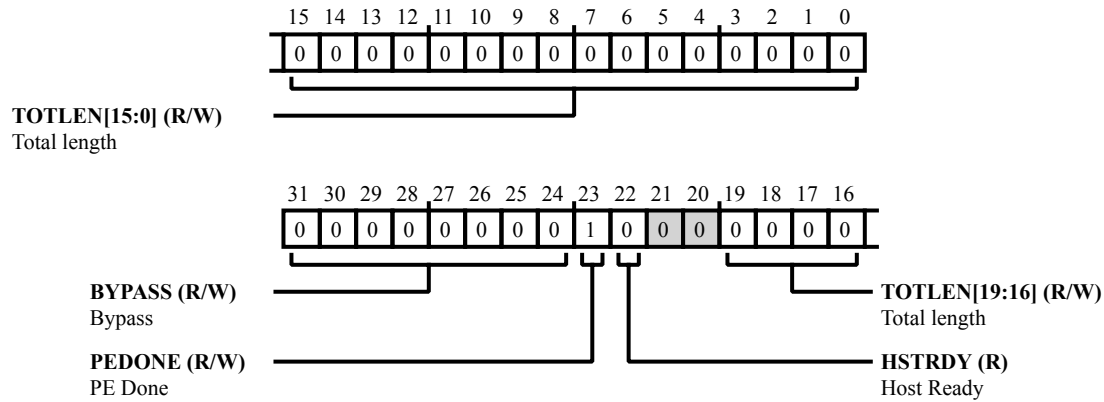


Figure 33-30: `PKTE_LEN` Register Diagram

Table 33-56: `PKTE_LEN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYPASS	Bypass. The <code>PKTE_LEN.BYPASS</code> bit field indicates the length of data in words that must bypass the packet engine and are directly copied from the source buffer to the destination buffer. The packet engine does not process this data. Valid bypass offsets range from 0 (0x00) to 255 (0xFF) words. For SRTP operations, this field specifies the offset in words between the hash and encrypt/decrypt data.
23 (R/W)	PEDONE	PE Done. The <code>PKTE_LEN.PEDONE</code> bit is a mirrored bit from the <code>PKTE_CTL_STAT.PERDY</code> bit. The bit is repeated here to guarantee ownership consistency between the first and last word. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.
22 (R/NW)	HSTRDY	Host Ready. The <code>PKTE_LEN.HSTRDY</code> bit is a mirrored bit of the <code>PKTE_CTL_STAT.HOSTRDY</code> bit. The bit is repeated here to guarantee ownership consistency between the first and last word. It should also be set along with the <code>PKTE_CTL_STAT.HOSTRDY</code> bit when the command descriptor is finished being populated. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.

Table 33-56: PKTE_LEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	TOTLEN	<p>Total length.</p> <p>Command Descriptor:</p> <p>The <code>PKTE_LEN.TOTLEN</code> bit field indicates the total length (in bytes) of all data to be passed to the packet engines input buffer for an operation. Exceptions are the PRNG init and PRNG generate operations. The PRNG init operation does not require any input data; this field must be zero.</p> <p>For the PRNG generate operation, this field indicates the number of pseudo-random bytes to be generated. Valid lengths range from 16 (0x00010) to $255 * 16 = 4080$ (0x00FF0) bytes in multiples of 16 bytes. Valid lengths for the basic operation range from 1 (0x00001) to 1,048,575 (0xFFFFF) bytes. This is the length of the data to be encrypted or hashed and includes the bypass data and padding bytes.</p> <p>Valid lengths for IPsec ESP range from 1 (0x00001) to 65535 (0xFFFFF) bytes. This is the length of the IP payload.</p> <p>Valid lengths for SSL v3.0, TLS v1.x and DTLS range from 1 (0x00001) to 16383 (0x03FFF). This is the length of the payload.</p> <p>Valid lengths for SRTP range from 1 (0x00001) to 65535 (0xFFFFF). This is the length of the payload.</p> <p>Note: A length of zero bytes is illegal and will result in an error status code in the result descriptor.</p> <p>Result Descriptor:</p> <p>Upon completion of an operation, the <code>PKTE_LEN.TOTLEN</code> field indicates the result length of the result packet. Valid lengths range from 1 (0x001) to 1,048,575 (0xFFFFF) bytes. This includes the bypass data and padding bytes.</p> <p>Note: When an extended error (<code>PKTE_CTL_STAT[18]=1</code>) is reported in the result descriptor and no packet data is processed, this field returns zero.</p>

Packet Engine Output Buffer Count Register

The `PKTE_OUTBUF_CNT` register provides the number of data bytes there are in the output buffer. The `PKTE_OUTBUF_CNT` register is used in direct host mode only.

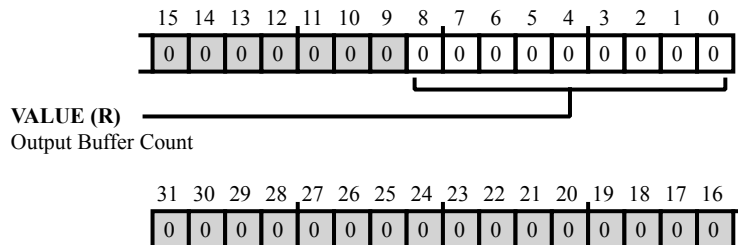


Figure 33-31: `PKTE_OUTBUF_CNT` Register Diagram

Table 33-57: `PKTE_OUTBUF_CNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Output Buffer Count. The <code>PKTE_OUTBUF_CNT.VALUE</code> bit field provides the number of bytes in the output buffer. The packet engine increments the counter by 4 when a 32-bit word is written to the output buffer.

Packet Engine Output Buffer Count Decrement Register

A host connected via the system slave bus can decrement the output buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The `PKTE_OUTBUF_DECR` register is used in direct host mode only.

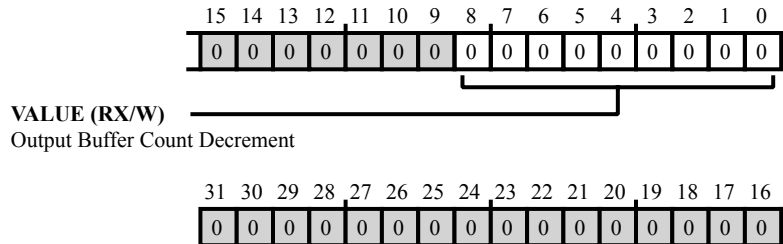


Figure 33-32: PKTE_OUTBUF_DECR Register Diagram

Table 33-58: PKTE_OUTBUF_DECR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Output Buffer Count Decrement. The <code>PKTE_OUTBUF_DECR.VALUE</code> bit field is the value written is subtracted to the output buffer counter. Valid values range from 4 to 256, in multiples of 4.

Packet Engine Result Descriptor Ring Base Address

The `PKTE_RDRBASE_ADDR` register holds the result descriptor ring base address in host memory. It is only applicable in autonomous ring mode and target command mode with RDR enabled. Note that in target command mode, the CDR is not used, but the RDR must be configured when enabled so that the packet engine knows where to write the result descriptors.

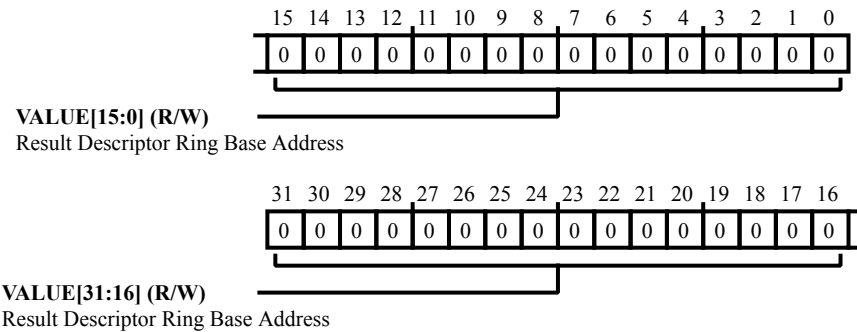


Figure 33-33: `PKTE_RDRBASE_ADDR` Register Diagram

Table 33-59: `PKTE_RDRBASE_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Result Descriptor Ring Base Address. The <code>PKTE_RDRBASE_ADDR.VALUE</code> bit field specifies the base location of the result descriptor ring in the host memory space.

Packet Engine Result Descriptor Count Registers

The `PKTE_RDSC_CNT` register holds the counter for the number of descriptors in the Result Descriptor Ring (RDR). It is incremented by the packet engine each time a valid result descriptor is written to the RDR.

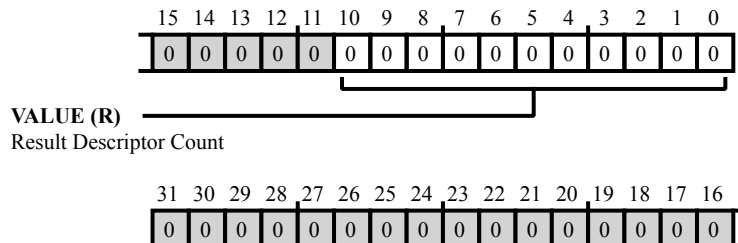


Figure 33-34: `PKTE_RDSC_CNT` Register Diagram

Table 33-60: `PKTE_RDSC_CNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Result Descriptor Count. The <code>PKTE_RDSC_CNT.VALUE</code> bit field provides the number of result descriptors in the result descriptor ring. The packet engine increments the counter when a valid result descriptor is written to the RDR.

Packet Engine Result Descriptor Count Decrement Registers

The `PKTE_RDSC_DECR` register is accessible by the host connected through the system slave bus can decrement the result descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

With an RDR enabled, this is the number of result descriptors that have been read by the host. With an RDR disabled, this indicates that the host has read one valid result descriptor.

In autonomous ring mode or target command mode with the RDR enabled, the host must process 1 to 255 result descriptors from the RDR and then write this register with the number of result descriptors that have been processed by the host.

In direct host mode or target command mode with the RDR disabled, the host must read one result descriptor from the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is read. An RDR threshold interrupt is activated when the result descriptor counter exceeds the threshold value set in the `PKTE_RING_THRESH` register. This interrupt can be used to wake up a process that stalled on an empty RDR.

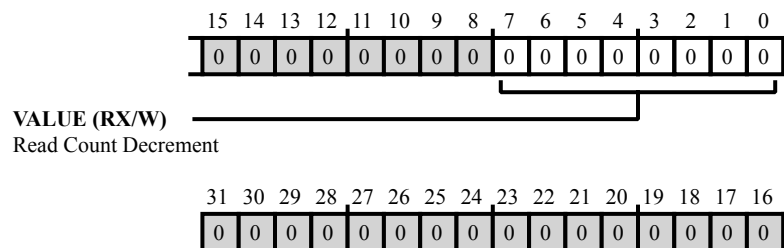


Figure 33-35: `PKTE_RDSC_DECR` Register Diagram

Table 33-61: `PKTE_RDSC_DECR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Read Count Decrement. The value written to the <code>PKTE_RDSC_DECR.VALUE</code> bit field is subtracted from the result descriptor counter. The counter is protected against underflow (See the <code>PKTE_RING_STAT</code> register). Note that bits [10:8] should be written with zeros.

Packet Engine Ring Configuration

The `PKTE_RING_CFG` register configures the size (in number of descriptor ring entries minus 1) for both the command descriptor ring and result descriptor ring in host memory. This register is only applicable for autonomous ring mode and target command mode with RDR enabled.

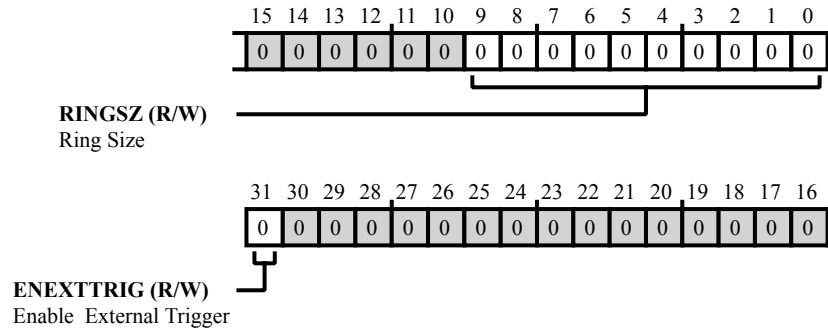


Figure 33-36: `PKTE_RING_CFG` Register Diagram

Table 33-62: `PKTE_RING_CFG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ENEXTTRIG	Enable External Trigger. The <code>PKTE_RING_CFG.ENEXTTRIG</code> signal enables the increment of the <code>PKTE_CDSC_CNT</code> register through the external input pin <code>ext_cd_cnt_incr</code> and enables the decrement of the <code>PKTE_RDSC_CNT</code> fields through the external input pin <code>ext_rd_cnt_decr</code> .
9:0 (R/W)	RINGSZ	Ring Size. The <code>PKTE_RING_CFG.RINGSZ</code> bit field specifies the size of the command ring in number of descriptors, minus 1. Valid sizes range from 1 (for 2 descriptors) to 1023 (for 1024 descriptors). The accompanying result ring will have the same size.

Packet Engine Ring Pointer Status

The `PKTE_RING_PTR` register holds the pointers to the current entry of the Command Descriptor Ring (CDR) and Result Descriptor Ring (RDR).

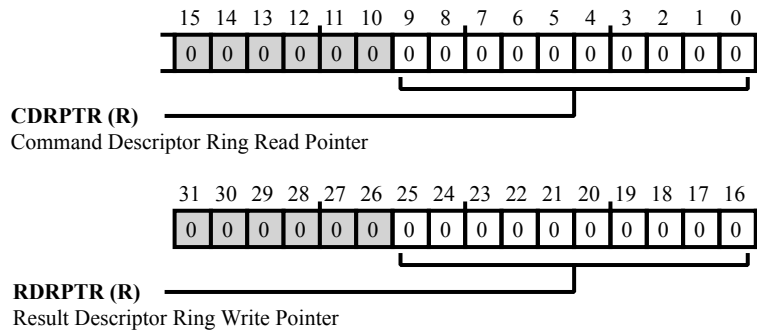


Figure 33-37: `PKTE_RING_PTR` Register Diagram

Table 33-63: `PKTE_RING_PTR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/NW)	RDRPTR	Result Descriptor Ring Write Pointer. The <code>PKTE_RING_PTR.RDRPTR</code> bit field indicates the entry number in the RDR that will be written next by the packet engine. The <code>PKTE_RING_PTR.RDRPTR</code> bit field is reset to zero after starting up and updated after every result descriptor write DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (<code>PKTE_RING_CFG.RINGSZ</code>) bit field.
9:0 (R/NW)	CDRPTR	Command Descriptor Ring Read Pointer. The <code>PKTE_RING_PTR.CDRPTR</code> bit field indicates the entry number in the CDR that will be read next by the packet engine. The <code>PKTE_RING_PTR.CDRPTR</code> bit field is reset to zero after starting up and updated after every command descriptor read DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (<code>PKTE_RING_CFG.RINGSZ</code>) field.

Packet Engine Ring Status

The `PKTE_RING_STAT` register gives indication of either a Command Descriptor Ring (CDR) overflow or a Result Descriptor Ring (RDR) underflow. A ring error (ringerr) interrupt in the interrupt controller is activated on a command descriptor ring overflow or a result descriptor ring underflow. This type of error can occur when the host and the packet engine get out-of-sync. The host can read this register to retrieve information on which ring is corrupted. The corrupted ring must be reset and reinitialized. See the `PKTE_CFG.RSTRING` bit.

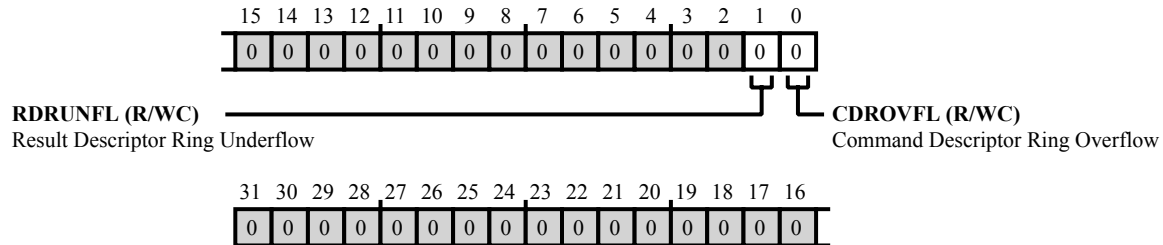


Figure 33-38: `PKTE_RING_STAT` Register Diagram

Table 33-64: `PKTE_RING_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/WC)	RDRUNFL	Result Descriptor Ring Underflow. The <code>PKTE_RING_STAT.RDRUNFL</code> bit is set when the command descriptor count (<code>PKTE_RDSC_CNT</code>) register is decremented below zero. This bit is reset with a write of any value.
0 (R/WC)	CDROVFL	Command Descriptor Ring Overflow. The <code>PKTE_RING_STAT.CDROVFL</code> bit is set when the command descriptor count (<code>PKTE_CDSC_CNT</code>) register is incremented above the ring size (<code>PKTE_RING_CFG.RINGSZ</code>) bits. This bit is reset with a write of any value.

Packet Engine Ring Threshold Registers

To reduce the amount of packet engine result interrupts, the `PKTE_RING_THRESH` register contains threshold and time-out values.

The CDR threshold (`cdrthrsh`) interrupt indicates that the command descriptor counter is less than or equal to the CDR threshold (`cdrthrsh`) value set in this register. This interrupt can be used to wake up a process that stalled on a full CDR.

The RDR threshold (`rdrthrsh`) interrupt indicates that the result descriptor counter exceeds the result descriptor threshold set here, or that the result descriptor counter is non-zero for a time longer than the result descriptor time-out setting. The RDR result interrupt remains active until the result descriptor counter is decremented below the RDR threshold (`rdrthrsh`) value. In case the interrupt is the result of a time-out and the result descriptor counter is below the threshold value, the result descriptor counter must be decremented once before the interrupt can be cleared in the interrupt controller.

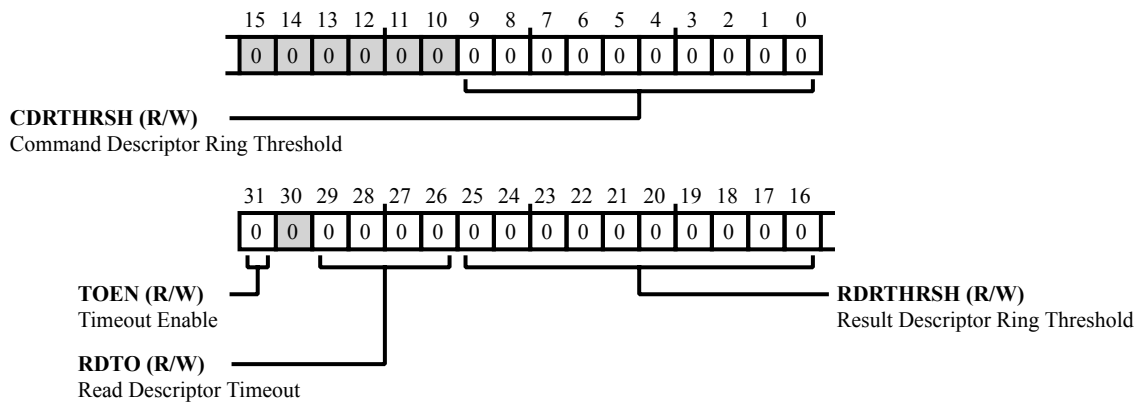


Figure 33-39: `PKTE_RING_THRESH` Register Diagram

Table 33-65: `PKTE_RING_THRESH` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TOEN	Timeout Enable. A 1 in the <code>PKTE_RING_THRESH</code> . <code>TOEN</code> bit indicates the result descriptor timeout counter is enabled. This bit can be used to de-activate the timeout counter to save power.

Table 33-65: PKTE_RING_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:26 (R/W)	RDTO	<p>Read Descriptor Timeout.</p> <p>The timeout enable (<code>PKTE_RING_THRESH.TOEN</code>) bit in this register must be set to activate this <code>PKTE_RING_THRESH.RDTO</code> result descriptor timeout counter. The <code>rdthrsh</code> interrupt activates when the RD counter for the RDR is non-zero for more than $2^{(N+10)}$ internal system clock cycles, where 'N' is the value set in this field. Valid settings range from 0 to 15. The minimum time-out value for N=0 is 1024 clock cycles and the maximum time-out value for N=15 is 33554432 clock cycles. At 100 MHz, this is 5.12 us for N=0 and 335.55 ms for N=15.</p> <p>Note: The time-out delay may not be exact - expect a variation on the order of 1024 system clock cycles (just more than one microsecond at 100 MHz system clock frequency).</p>
25:16 (R/W)	RDRTHRSH	<p>Result Descriptor Ring Threshold.</p> <p>The <code>rdthrsh</code> interrupt activates when the RD counter for the RDR exceeds the value set in the <code>PKTE_RING_THRESH.RDRTHRSH</code> field. Valid settings range from 0 to 1023.</p>
9:0 (R/W)	CDRTHRSH	<p>Command Descriptor Ring Threshold.</p> <p>The <code>cdthrsh</code> interrupt activates when CD counter for the CDR is below or equal the value set in the <code>PKTE_RING_THRESH.CDRTHRSH</code> field. Valid settings range from 0 to 1023.</p>

Packet Engine SA Address

The `PKTE_SA_ADDR` register holds the start address of the SA record.

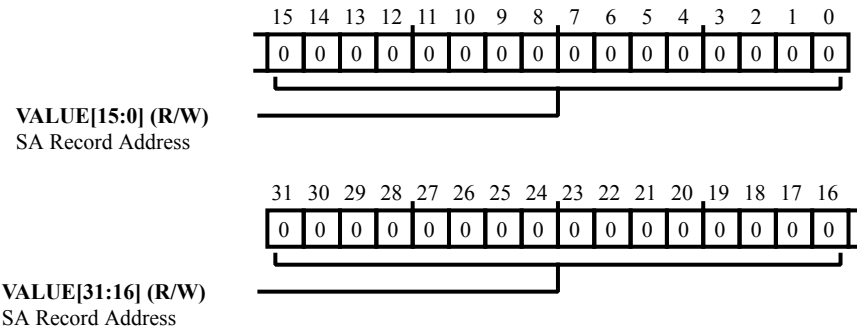


Figure 33-40: `PKTE_SA_ADDR` Register Diagram

Table 33-66: `PKTE_SA_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Record Address. The <code>PKTE_SA_ADDR.VALUE</code> bit field holds the start address of the SA record.

ARC4 i and j Pointer Register

When starting a new ARC4 operation the `PKTE_SA_ARC4IJPTR` register contains the initialization value, which is zeros. After processing the ARC4 algorithm it contains the latest status of the `ARC4_IJ_PNTR`.

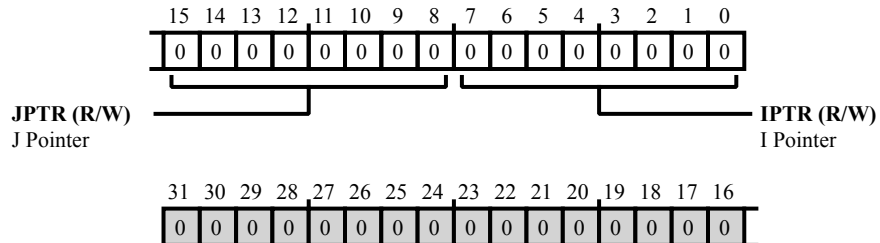


Figure 33-41: `PKTE_SA_ARC4IJPTR` Register Diagram

Table 33-67: `PKTE_SA_ARC4IJPTR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	JPTR	J Pointer. The <code>PKTE_SA_ARC4IJPTR</code> . JPTR bit field contains the j pointer into s-box array for swapping bytes with i pointer.
7:0 (R/W)	IPTR	I Pointer. The <code>PKTE_SA_ARC4IJPTR</code> . IPTR bit field contains the i pointer into s-box array for swapping bytes with j pointer.

SA Command 0

The two SA command registers, `PKTE_SA_CMD0` and `PKTE_SA_CMD1`, are used to control the cryptographic operation of the packet engine. The `PKTE_SA_CMD0` register contains the major control bits to define an operation while the `PKTE_SA_CMD1` register contains the minor control bits. In direct host mode, this is a write-only register.

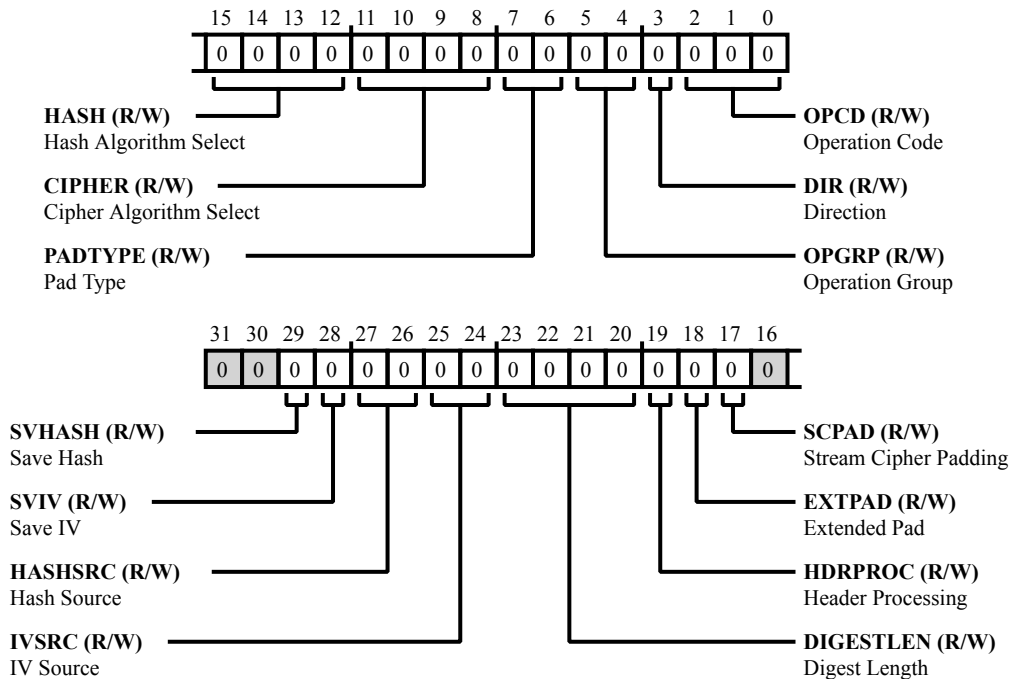


Figure 33-42: `PKTE_SA_CMD0` Register Diagram

Table 33-68: `PKTE_SA_CMD0` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	SVHASH	Save Hash. The <code>PKTE_SA_CMD0.SVHASH</code> bit indicates that the Hash State is saved to the <code>STATE_BYTE_CNT_X</code> and <code>STATE_IDIGEST_X</code> fields in the SA record in memory after completion of a crypto operation.
		0 Hash state is not saved
		1 Hash state is saved

Table 33-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	SVIV	Save IV. The PKTE_SA_CMD0 . SVIV bit field indicates that for DES or the AES the Initialization Vector (IV) is saved to the STATE_IV_X fields in the state record , or the ARC4 state is saved to the ARC4 state record, after completion of the crypto operation.
		0 ARC4 State is not saved.
		1 ARC4 State is not saved.
27:26 (R/W)	HASHSRC	Hash Source. The PKTE_SA_CMD0 . HASHSRC bit field selects the source of the hash digest used by the algorithm.
		0 From SA. Digest only hash byte count is forced to 0x40.
		1 Reserved
		2 From State. Read saved inner hash digest and saved hash byte count.
		3 No Load. Use the hash algorithm defined constants for the initial hash. Hash byte count is 0x00.
25:24 (R/W)	IVSRC	IV Source. The PKTE_SA_CMD0 . IVSRC bit field selects the source of the initialization vector used by the crypto algorithm.
		0 No load. Use previous result IV, not applicable for inbound data. This option should never be used for operations with DES-CBC or AES-CBC, (see RFC3602) or any AES counter modes
		1 From input buffer. The IV is provided as part of the input data stream.
		2 From State. Read STATE_IV_X, from the SA structure. Refer to inner hash digest register structure. Useful for resume operations.
		3 From internal PRNG. Not applicable for inbound operations.
23:20 (R/W)	DIGESTLEN	Digest Length. The PKTE_SA_CMD0 . DIGESTLEN bit field defines the length of the hash digest in words as put in the output buffer.
		0 3 Words (96-bit output)
		1 1 Word
		2 2 Words

Table 33-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		3 3 Words (IPsec)
		4 4 Words (MD5 and AES-based hash)
		5 5 Words (SHA-1)
		6 6 Words
		7 7 Words (SHA-224)
		8 8 Words (SHA-256)
		9 Reserved
		10 10 bytes (SRTP and TLS)
		11-15 Reserved
19 (R/W)	HDRPROC	Header Processing. The <code>PKTE_SA_CMD0.HDRPROC</code> bit enables header processing for protocol operations. There is no header-processing support for basic SSL, basic TLS and SRTP protocol operations as defined in the protocol group (see the Crypto and Hash Algorithms section). This bit must be zero for these operations; however, the protocol header must be supplied to the packet engine since it is part of the hash calculation. Refer to the protocol specifications for more information about header-processing support for a protocol.
		0 No header processing
		1 Header processing; insert the protocol header for out-bound operations, verify the protocol header for in-bound operations.
18 (R/W)	EXTPAD	Extended Pad. The <code>PKTE_SA_CMD0.EXTPAD</code> bit extends the number of padding types. Used in combination with <code>PKTE_SA_CMD0.PADTYPE</code> .
17 (R/W)	SCPAD	Stream Cipher Padding. The <code>PKTE_SA_CMD0.SCPAD</code> bit enables padding for stream ciphers algorithms.
15:12 (R/W)	HASH	Hash Algorithm Select. The <code>PKTE_SA_CMD0.HASH</code> bit field selects the hash algorithm.
		0 MD5
		1 SHA-1
		2 SHA-224
		3 SHA-256
		4-14 Reserved
		15 Null

Table 33-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	CIPHER	Cipher Algorithm Select. The <code>PKTE_SA_CMD0.CIPHER</code> bit field selects the cipher algorithm to be used for encryption and decryption. Note: Each type of protocol operation supports different sets of crypto algorithms. Refer to the Crypto and Hash Algorithms general processing section for details of the supported algorithms.
		0 DES
		1 Triple-DES
		2 ARC4
		3 AES
		4-14 Reserved
		15 Null
7:6 (R/W)	PADTYPE	Pad Type. The <code>PKTE_SA_CMD0.PADTYPE</code> bit field indicates the type of crypto that must be generated for outbound packets or checked for inbound packets.
		0 Select IPsec operation (if Bit 18=0); Reserved (if Bit 18=1)
		1 PKCS#7 (if Bit 18=0); Select TLS/DTLS Pad, required for TLS/DTLS operation (if Bit 18=1)
		2 Constant pad (if Bit 18=0); Select Constant SSL Pad, required for SSL operation (if Bit 18=1)
		3 Zero pad (if Bit 18=0), Reserved (if Bit 18=1)
5:4 (R/W)	OPGRP	Operation Group. The <code>PKTE_SA_CMD0.OPGRP</code> bit field defines the operation groups. Refer to the Basic Operations and Decoding section for more information.
		0 Basic operation group
		1 Protocol operation group
		2 Extended protocol operations group
		3 Reserved
3 (R/W)	DIR	Direction. The <code>PKTE_SA_CMD0.DIR</code> bit field selects the direction of operation.
		0 Outbound operations
		1 Inbound operations

Table 33-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	OPCD	Operation Code. The PKTE_SA_CMD0.OPCD bit field selects the operation within the operation group.

SA Command 1

The `PKTE_SA_CMD1` register contains the minor control bits that define an operation. In direct host mode, this is a write-only register.

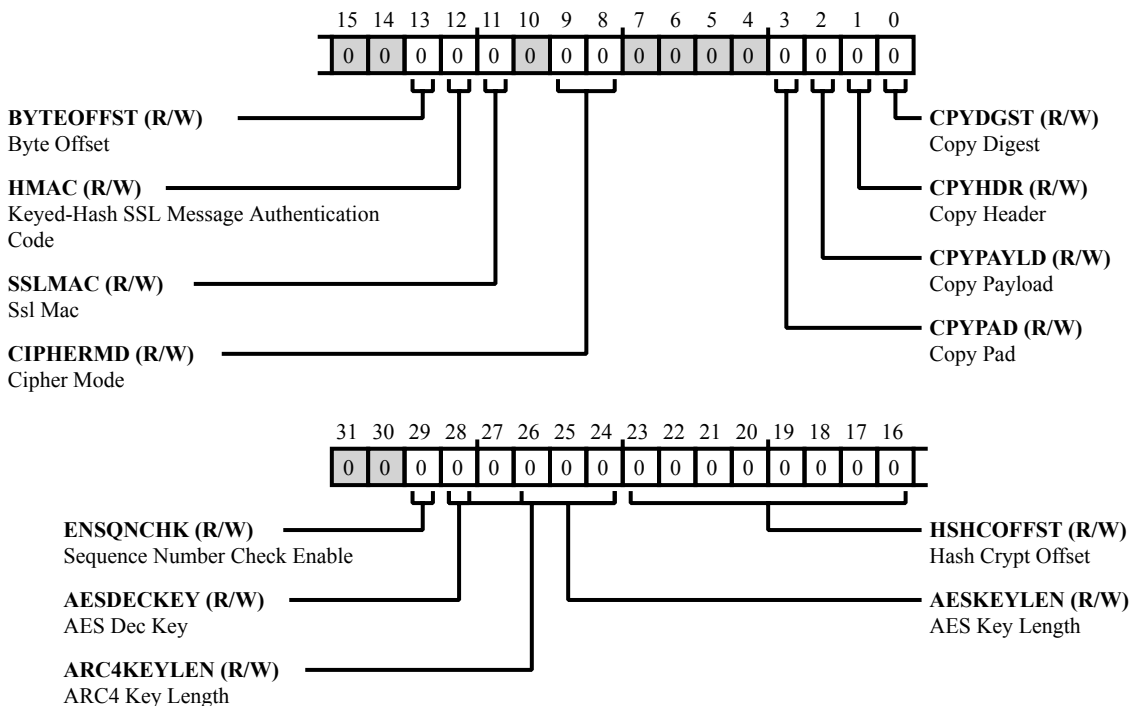


Figure 33-43: PKTE_SA_CMD1 Register Diagram

Table 33-69: PKTE_SA_CMD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	ENSQNCHK	Sequence Number Check Enable. The <code>PKTE_SA_CMD1.ENSQNCHK</code> bit defines that the key in the SA key field is an AES encrypt key or an AES decrypt key.
		0 Disable sequence number check
		1 Enable sequence number check
28 (R/W)	AESDECKEY	AES Dec Key. If the <code>PKTE_SA_CMD1.AESDECKEY</code> bit is set, the key in loaded in the <code>PKTE_SA_KEY[n]</code> registers are expected to be the key from the last round from key expansion. If not set, the key loaded in the <code>PKTE_SA_KEY[n]</code> registers are expected to be the same key used during the encryption process.
		0 AES key is an encrypt key.
		1 AES key is a decrypt key.

Table 33-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26:24 (R/W)	AESKEYLEN	AES Key Length. The PKTE_SA_CMD1.AESKEYLEN bit field select the size of the key used for the AES algorithm in increments of 64 bits.
		0-1 Reserved
		2 128 Bits
		3 192 Bits
		4 256 Bits
		5-7 Reserved
28:24 (R/W)	ARC4KEYLEN	ARC4 Key Length.
23:16 (R/W)	HSHCOFFST	<p>Hash Crypt Offset.</p> <p>For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the PKTE_SA_CMD1.HSHCOFFST bit field specifies the offset between the hash data and the encrypt/decrypt data. The data to be hashed is assumed to come first, with an offset to the beginning of encrypt/decrypt data.</p> <p>When PKTE_SA_CMD1.BYTEOFFST, bit 13, is zero, then the offset is defined in 32-bit words. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 0 to 255.</p> <p>When PKTE_SA_CMD1.BYTEOFFST, bit 13, is one, then the offset is defined in 8-bit bytes. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 4 to 255. (The IV size is two words for DES, Triple-DES and AES-CTR and four words for AES-CBC and AES-ICM operations).</p> <p>Other operations do not use these bits (a default value is applied by the packet engine).</p>
13 (R/W)	BYTEOFFST	Byte Offset. The PKTE_SA_CMD1.BYTEOFFST bit defines how the PKTE_SA_CMD1.HSHCOFFST, bits of this register are used.
		0 HASH_CRYPT_OFFSET is defined in 32-bit words
		1 HASH_CRYPT_OFFSET is defined in 8-bit bytes
12 (R/W)	HMAC	Keyed-Hash SSL Message Authentication Code. For basic operations that include hashing, the PKTE_SA_CMD1.HMAC bit enables the HMAC processing, which calls for an extra outer hash operation.
		0 Standard Hash
		1 HMAC Processing

Table 33-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	SSLMAC	Ssl Mac.	
		0	Standard Hash
		1	SSL-MAC processing
9:8 (R/W)	CIPHERMD	Cipher Mode. The <code>PKTE_SA_CMD1.CIPHERMD</code> bit field selects the crypto mode to be used for the cipher algorithm.	
		0	Electronic Code Book (ECB) used for DES and AES
		1	Cipher Block Chaining (CBC) used for DES and AES
		2	AES Counter Mode (CTR) for IPsec using a 32-bit counter
3 (R/W)	CPYPAD	Copy Pad. The <code>PKTE_SA_CMD1.CPYPAD</code> bit indicates that the padding data for an inbound operation is copied to the output buffer and saved in memory.	
		0	Do not copy the padding to output
		1	Copy padding to output
2 (R/W)	CPYPAYLD	Copy Payload. The <code>PKTE_SA_CMD1.CPYPAYLD</code> bit indicates that the payload data is copied to the output buffer and saved in memory.	
		0	Do not copy the payload to output
		1	Copy payload to output
1 (R/W)	CPYHDR	Copy Header. The <code>PKTE_SA_CMD1.CPYHDR</code> bit indicates that the protocol header is copied to the output buffer and saved in memory. For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the header is defined as the Hash/Crypt Offset data (authenticated only).	
		0	Do not copy the header to output
		1	Copy header to output

Table 33-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	CPYDGST	Copy Digest. The PKTE_SA_CMD1 . CPYDGST bit copies the hash result is to the output buffer and saves in memory. The length of the hash result is defined by the PKTE_SA_CMD0 . DIGESTLEN field.
		0 Do not copy hash result to output
		1 Copy hash result to output, when the command descriptor PKTE_CTL_STAT . HASHFINAL bit is set.

SA Inner Hash Digest Registers

The `PKTE_SA_IDIGEST[n]` registers are a set of eight 32-bit read/write registers.

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers are used to enter a start hash state, and to read the interim or final hash digest.

For IPsec, TLS and DTLS operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with basic hash or HMAC authentication with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the pre-computed inner hash digest. This is the hash of the hash-key padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the inner hash pre-compute; this is the hash of the `MAC_WRITE_SECRET` padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the `MAC_WRITE_SECRET`. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the hash-key pad (0x36 hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

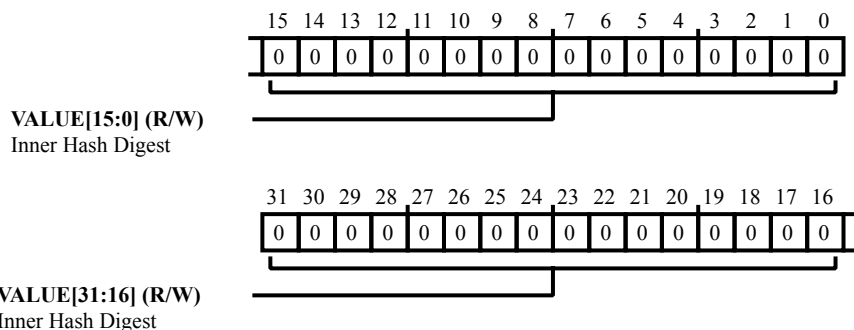


Figure 33-44: `PKTE_SA_IDIGEST[n]` Register Diagram

Table 33-70: `PKTE_SA_IDIGEST[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Hash Digest.

SA Key Registers

These are the `PKTE_SA_KEY[n]` registers for DES, Triple-DES, ARC4 and AES: A set of eight 32-bit write only registers. The reset value of these registers is zero.

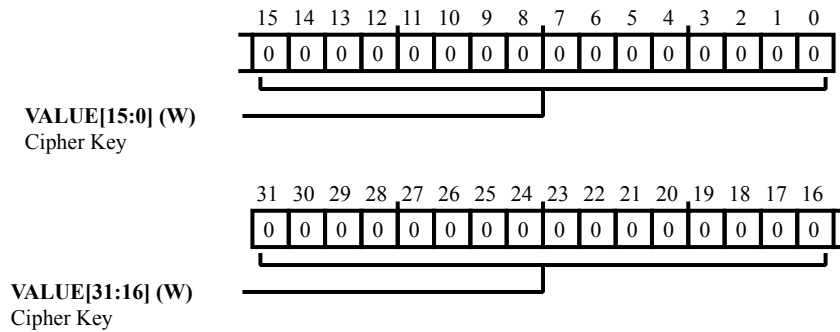


Figure 33-45: `PKTE_SA_KEY[n]` Register Diagram

Table 33-71: `PKTE_SA_KEY[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Cipher Key.

SA Initialization Vector Register

The `PKTE_SA_NONCE` register is used for operations that make use of the IV value loaded from the SA record. This register is used both to enter a starting IV state, as well as for reading the interim or final IV. For IPsec out-bound operations, it is recommended that the automatic IV insertion mode be used, this register is not needed. For IPsec inbound operations, the IV is extracted from the header of the packet.

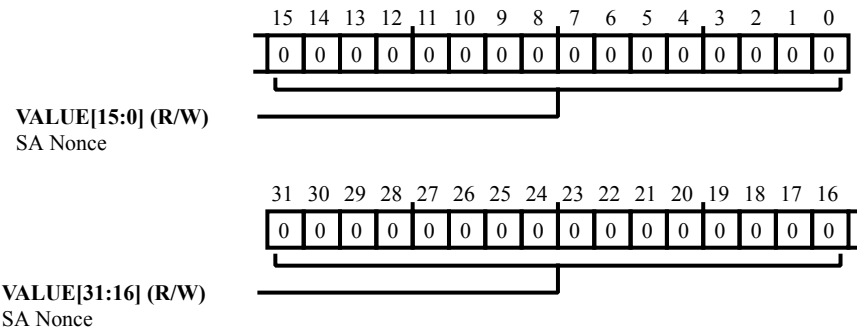


Figure 33-46: PKTE_SA_NONCE Register Diagram

Table 33-72: PKTE_SA_NONCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Nonce.

SA Outer Hash Digest Registers

The `PKTE_SA_ODIGEST[n]` registers are a set of five eight 32-bit write-only registers.

For write operations, these registers contain the pre-computed outer hash digest for IPsec operations with basic HMAC operations with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA).

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers hold a start hash state, or the interim outer hash digest. They are only used for HMAC processing.

For IPsec, SSL, TLS, DTLS and SRTP operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with HMAC authentication with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the pre-computed outer hash digest. This is the hash of the hash-key padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the outer hash pre-compute; this is the hash of the `MAC_WRITE_SECRET` padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the `MAC_WRITE_SECRET`. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the required hash-key pad (0x5C hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

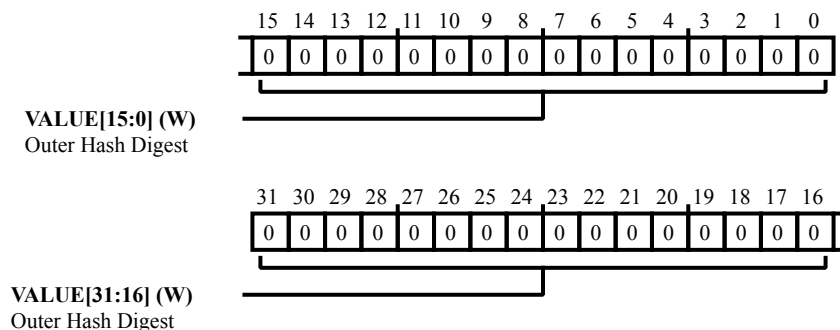


Figure 33-47: PKTE_SA_ODIGEST[n] Register Diagram

Table 33-73: PKTE_SA_ODIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Outer Hash Digest.

SA Ready Indicator

In direct host mode, a write to the `PKTE_SA_RDY` register triggers the packet engine to start processing using the command descriptor, SA record and state record in the packet engine registers. This register **MUST** be written for all direct host mode packet operations. It is intended that this register is written in sequence; as the entire SA record is written.

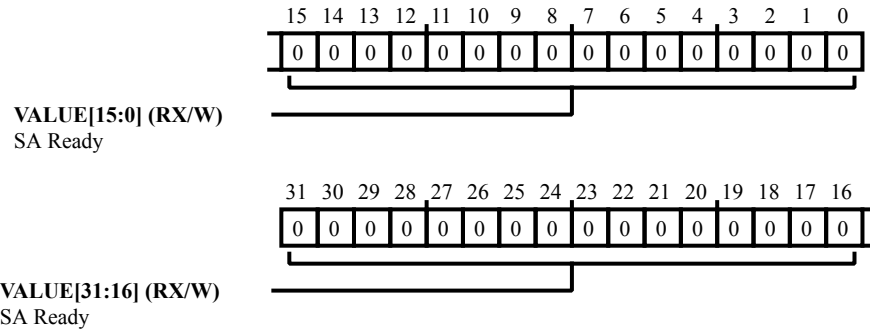


Figure 33-48: `PKTE_SA_RDY` Register Diagram

Table 33-74: `PKTE_SA_RDY` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	SA Ready.

SA Sequence Number Register

The `PKTE_SA_SEQNUM[n]` registers are a set of two read/write registers and are used for IPsec ESP, SSL, TLS, DTLS operations to specify the anti-replay sequence number value that is to be placed in the ESP header (outbound), or to be checked against for inbound packets. The packet engine manages this counter value for both inbound and outbound operations.

Outbound: The host writes the counter value stored in the SA record to this register to start an IPsec, SSL, TLS, DTLS operation. The packet engine automatically increments the count if header processing is selected. Upon successful completion, the host reads back this value and writes it to the SA record.

Inbound: The host writes the counter value stored in the SA record to this register to start an IPsec or DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the expected count value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Note: The description is only for the direct host mode. The sequence number for autonomous ring mode and target command mode are updated by the packet engine.

The reset value of this register is zero.

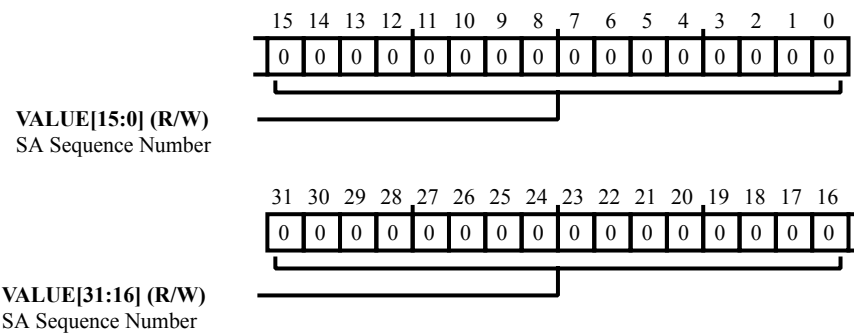


Figure 33-49: PKTE_SA_SEQNUM[n] Register Diagram

Table 33-75: PKTE_SA_SEQNUM[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA Sequence Number Mask Registers

The `PKTE_SA_SEQNUM_MSK[n]` registers are a set of two read/write registers and are used for IPsec ESP and DTLS operations to specify the anti-replay sequence number mask value for inbound operations. The packet engine manages this counter value automatically.

Inbound: The host writes the counter value stored in the SA record into this register upon starting an IPsec, DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the new mask value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Outbound: not used.

Note that the above description only applies to the direct host mode, for autonomous ring mode and target command mode the packet engine extracts the sequence number mask from the SA record.

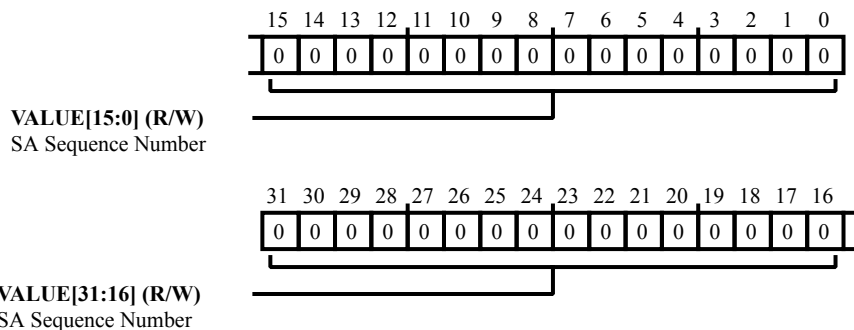


Figure 33-50: `PKTE_SA_SEQNUM_MSK[n]` Register Diagram

Table 33-76: `PKTE_SA_SEQNUM_MSK[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA SPI Register

For IPsec operations, the `PKTE_SA_SPI` register is written with the SPI (Security Parameters Index) associated with the inbound or outbound flow.

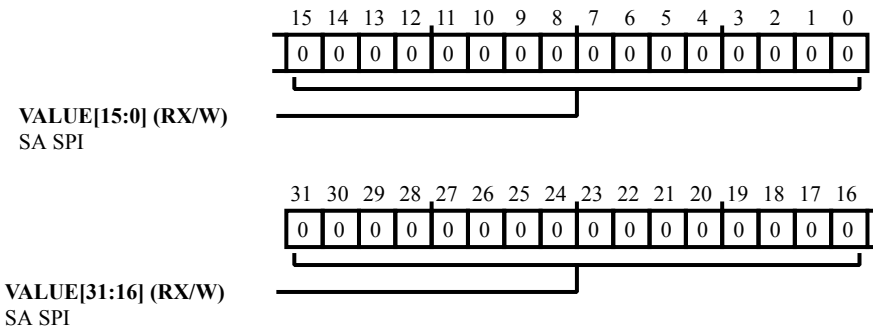


Figure 33-51: `PKTE_SA_SPI` Register Diagram

Table 33-77: `PKTE_SA_SPI` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	<p>SA SPI.</p> <p>The <code>PKTE_SA_SPI.VALUE</code> bit field is used for IPsec ESP operations to specify the Security Parameters Index (SPI) value that is to be placed in the ESP header. There is no need to read back this value at the end of an operation, since the Packet Engine does not change it.</p> <p>For SSL, TLS and DTLS this register stores the 8-bit TYPE field in bits [23:16], and the 16-bit Version field in bits [15:0] that are part of the protocol header.</p>

Packet Engine Source Address

The `PKTE_SRC_ADDR` register holds the starting (byte) address for the packet to be processed.

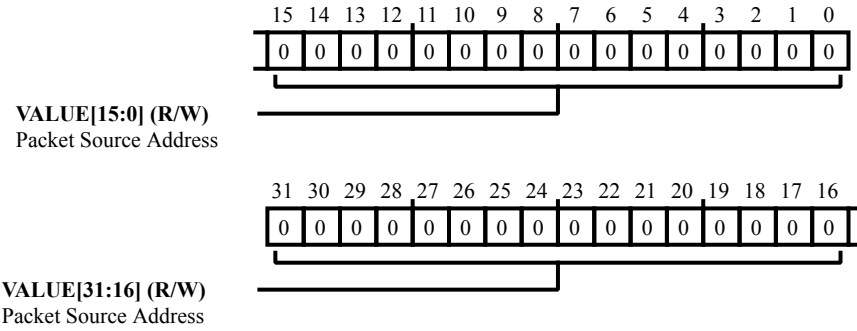


Figure 33-52: `PKTE_SRC_ADDR` Register Diagram

Table 33-78: `PKTE_SRC_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Source Address. The <code>PKTE_SRC_ADDR.VALUE</code> bit field holds the starting (byte) address for the packet to be processed.

Packet Engine Status Register

The `PKTE_STAT` register is used to provide the status of the packet engine. This register is useful in the direct host mode to determine when data must be written to or read from the packet engine, or for debugging the software when errors occur. This register can be ignored in autonomous ring mode and target command mode where the DMA engine controls the packet data I/O. This is a read-only register. A write to any of the bits has no effect.

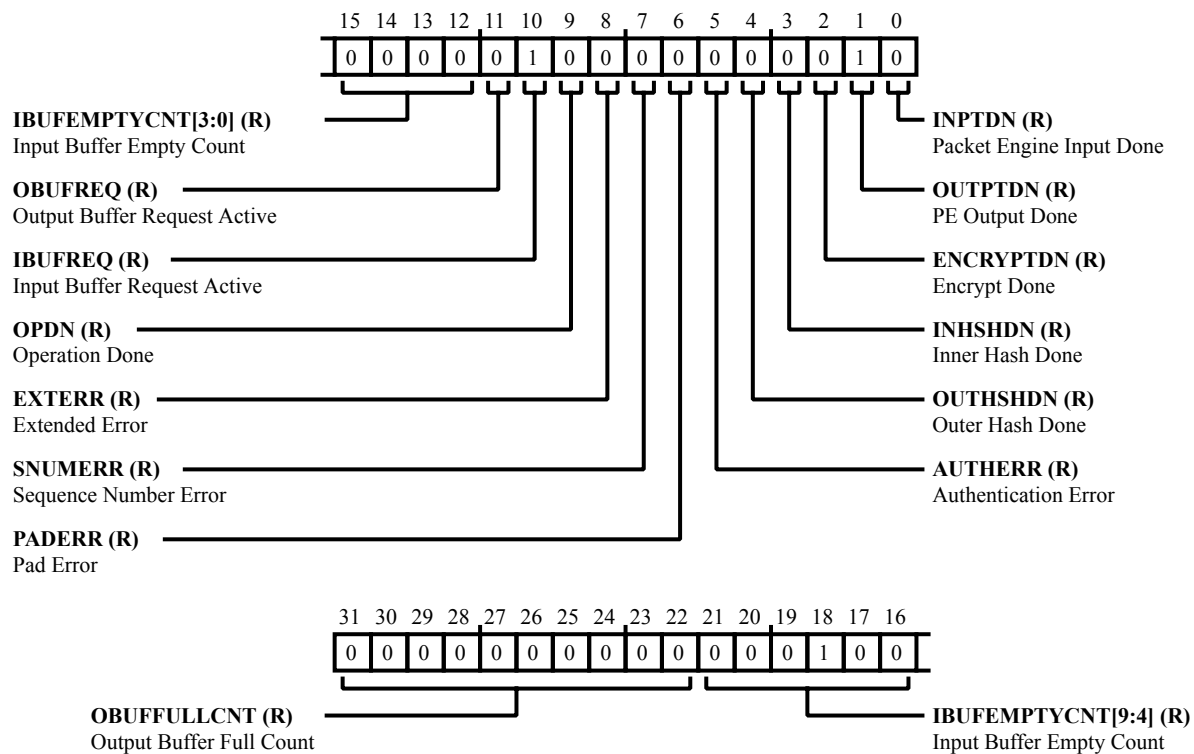


Figure 33-53: `PKTE_STAT` Register Diagram

Table 33-79: `PKTE_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:22 (R/NW)	<code>OBUFFULLCNT</code>	Output Buffer Full Count. The <code>PKTE_STAT.OBUFFULLCNT</code> bit field indicates the number of 32-bit words that are available in the packet engine output buffer. It works in conjunction with bit 11 from this register. When bit 11 is asserted, to indicate a request for output, the word count matches the specified output buffer threshold setting in the <code>PKTE_BUF_THRESH</code> register. For the last output for a given packet, any value from 1 dword to the full output buffer threshold can be seen. Transfers must be a multiple of full dwords. The application must read the <code>PKTE_LEN</code> field in the result descriptor to determine the exact byte-length of the result.

Table 33-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration		
21:12 (R/NW)	IBUFEMPTYCNT	<p>Input Buffer Empty Count.</p> <p>The <code>PKTE_STAT.IBUFEMPTYCNT</code> bit field indicates the number of 32-bit empty spaces that are available in the packet engine input buffer. It works in conjunction with the <code>PKTE_STAT.IBUFREQ</code> bit (10) from this register.</p> <p>The value in the register is deducted from the specified packet length, so will never exceed the number of dwords that remain in the packet. For packets smaller than the buffer size, this register typically indicates that buffer space is available for the entire packet (rounded up to the nearest dword). For very large packets, these bits usually have a value around the maximum buffer size, indicating that the full input buffer is available.</p>		
11 (R/NW)	OBUFREQ	Output Buffer Request Active.		
		The <code>PKTE_STAT.OBUFREQ</code> bit indicates that the packet engine requests output data to be read from the output buffer.		
		<table border="1"> <tr> <td>0</td> <td>No request for output data</td> </tr> <tr> <td>1</td> <td>Request for output data</td> </tr> </table>	0	No request for output data
0	No request for output data			
1	Request for output data			
10 (R/NW)	IBUFREQ	Input Buffer Request Active.		
		The <code>PKTE_STAT.IBUFREQ</code> bit indicates that the packet engine requests input data to be written to the input buffer.		
		<table border="1"> <tr> <td>0</td> <td>No request for input data</td> </tr> <tr> <td>1</td> <td>Request for input data</td> </tr> </table>	0	No request for input data
0	No request for input data			
1	Request for input data			
9 (R/NW)	OPDN	Operation Done.		
		The <code>PKTE_STAT.OPDN</code> bit indicates that the packet engine has finished processing a packet when in direct host mode. This bit is zero in autonomous ring mode and target command mode.		
		<table border="1"> <tr> <td>0</td> <td>Packet engine is idle</td> </tr> <tr> <td>1</td> <td>Packet engine has finished processing a packet</td> </tr> </table>	0	Packet engine is idle
0	Packet engine is idle			
1	Packet engine has finished processing a packet			
8 (R/NW)	EXTERR	Extended Error.		
		The <code>PKTE_STAT.EXTERR</code> bit indicates that an extended error occurred for this packet. For more information, refer to table Extended Error Codes - Status Encoding.		
		<table border="1"> <tr> <td>0</td> <td>No extended error</td> </tr> <tr> <td>1</td> <td>Extended error</td> </tr> </table>	0	No extended error
0	No extended error			
1	Extended error			

Table 33-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	SNUMERR	Sequence Number Error. For an inbound operation, the PKTE_STAT . SNUMERR bit indicates that there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition. For more information, refer to table Extended Error Codes - Status Encoding.
		0 No sequence number error
		1 Input done, all bytes written to input buffer
6 (R/NW)	PADERR	Pad Error. The PKTE_STAT . PADERR bit indicates that an inbound crypto pad fault is detected. For more information about pad verification, refer to the Pad Verification and Consumption section.
		0 No pad error
		1 Pad error
5 (R/NW)	AUTHERR	Authentication Error. The PKTE_STAT . AUTHERR bit indicates that an inbound ICV (for IPsec) or TAG (for SRTP) or MAC (for SSL/TLS/DTLS) fault is detected; the value carried within the packet did not match the value just computed.
		0 No authentication error
		1 Authentication error
4 (R/NW)	OUTHSHDN	Outer Hash Done. The PKTE_STAT . OUTHSHDN bit indicates that the outer hash processing for this packet is finished.
		0 Outer hash busy
		1 Outer hash done
3 (R/NW)	INHSHDN	Inner Hash Done. The PKTE_STAT . INHSHDN bit indicates that the inner hash processing for this packet is finished.
		0 Inner hash busy
		1 Inner hash done
2 (R/NW)	ENCRYPTDN	Encrypt Done. The PKTE_STAT . ENCRYPTDN bit indicates that the encryption or decryption for this packet is finished.
		0 Encryption or decryption busy
		1 Encryption or decryption done

Table 33-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	OUTPTDN	PE Output Done. The <code>PKTE_STAT.OUTPTDN</code> bit indicates that the output data for the current packet is read from the packet engine output buffer.
		0 Output not done, more output bytes available
		1 Output done, all bytes read from the output buffer
0 (R/NW)	INPTDN	Packet Engine Input Done. The <code>PKTE_STAT.INPTDN</code> bit indicates that the number of bytes specified in the command descriptor <code>PKTE_LEN</code> field is written into the packet engine input buffer.
		0 Input not done, more input bytes expected
		1 Input done, all bytes written to input buffer

Packet Engine State Record Address

The `PKTE_STATE_ADDR` register holds the start address of the SA state record.

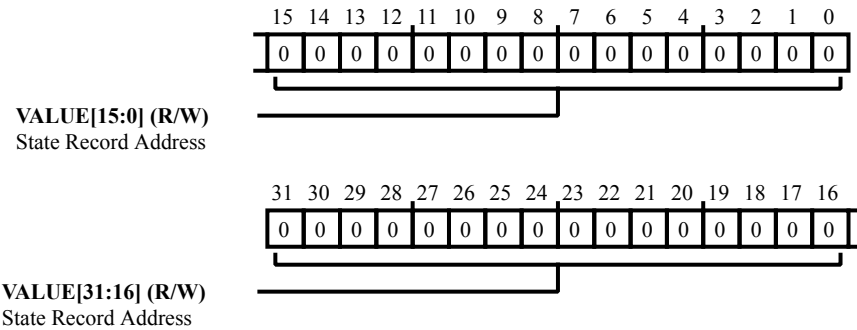


Figure 33-54: `PKTE_STATE_ADDR` Register Diagram

Table 33-80: `PKTE_STATE_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Record Address. The <code>PKTE_STATE_ADDR.VALUE</code> bit field holds the start address of the SA state record.

State Hash Byte Count Registers

The `PKTE_STATE_BYTE_CNT[n]` registers are used to enter a starting hash byte count, as well as to read the interim or final byte count.

For some hash operations, these registers are ignored and the byte count is internally set to 64 (0x40 hex) to indicate that the first 64 bytes (512 bits) hash block has been processed using a pre-computed hash state. These operations are:

All IPsec, SSL, TLS, DTLS and SRTP operations that use authentication; the "pre-computed" inner and outer hash digests are loaded from SA words 10 - 19.

Basic operations with `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA) specified. For Basic Hash with no HMAC, a pre-computed digest is loaded from SA words 10 - 14. For Basic Hash with HMAC, the inner and outer digests are loaded from SA words 10 - 19.

Note: Protocol operations can not be suspended in mid-packet and resumed later, therefore protocol operations do not use these registers.

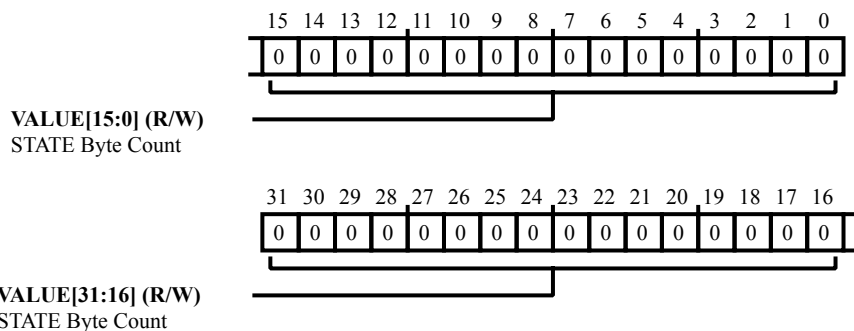


Figure 33-55: PKTE_STATE_BYTE_CNT[n] Register Diagram

Table 33-81: PKTE_STATE_BYTE_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	STATE Byte Count.

State Inner Digest Registers

The `PKTE_STATE_IDIGEST[n]` registers consist of eight 32-bit registers. These read/write registers are used to read the interim or final hash digest. The `PKTE_STATE_IDIGEST[n]` registers are only used with basic operations involving basic hash, and are typically used for operations that must be suspended and resumed in the middle of a hash. The interim hash state can be read from these registers along with the hash byte-count from the previous register. Both can be restored when resuming the hash. The appropriate save hash state (`PKTE_SA_CMD0.SVHASH=1`) and load hash from state (`PKTE_SA_CMD0.HASHSRC=0b10`) settings must be used. These registers are a mirror of the SA record inner hash digest register.

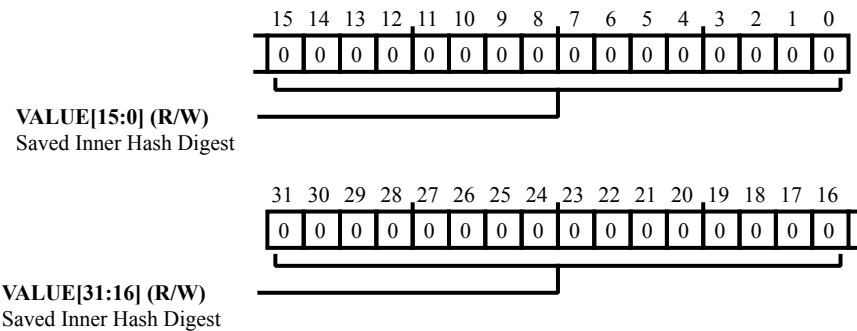


Figure 33-56: `PKTE_STATE_IDIGEST[n]` Register Diagram

Table 33-82: `PKTE_STATE_IDIGEST[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Saved Inner Hash Digest.

State Initialization Vector Registers

The `PKTE_STATE_IV[n]` consists of four 32-bit registers. These registers are used to enter a starting IV state and to read the interim or final IV. `PKTE_STATE_IV0` and `PKTE_STATE_IV1` are used with DES/3DES cipher while `PKTE_STATE_IV0` to `PKTE_STATE_IV3` are used with AES cipher. The reset value of these registers is zero.

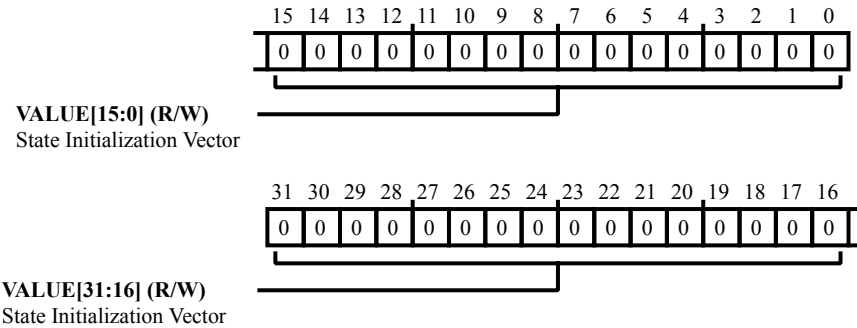


Figure 33-57: `PKTE_STATE_IV[n]` Register Diagram

Table 33-83: `PKTE_STATE_IV[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Initialization Vector. The <code>PKTE_STATE_IV[n].VALUE</code> bit field is used to enter a starting IV state and to read the interim or final IV.

Packet Engine User ID

The `PKTE_USERID` register is a read/write register that gives identification to a command descriptor and the resultant result descriptor. The host is free to use this field for its own purpose. The host can write a unique identifier to the register in direct host mode or includes it as part of the command descriptor in autonomous ring mode. The `PKTE_USERID` register value passes through the packet engine without alteration to the result descriptor to be read back by the host.

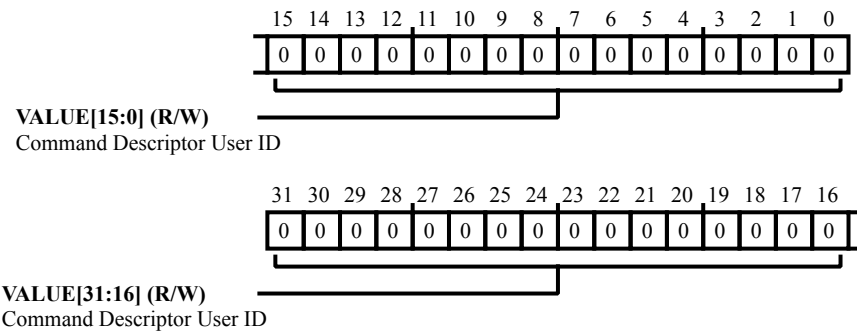


Figure 33-58: `PKTE_USERID` Register Diagram

Table 33-84: `PKTE_USERID` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor User ID. The <code>PKTE_USERID.VALUE</code> bit field gives identification to a command descriptor and the resultant result descriptor.

34 Public Key Accelerator (PKA)

The PKA helps offload computationally-intensive operations commonly found in public key cryptography algorithms.

PKA Features

The PKA engine provides the following basic operations:

- Large vector addition, subtraction, and combined addition/subtraction
- Large vector shift right or left
- Large vector multiplication, division (with and without quotient)
- Large vector compare and copy

The PKA engine provides the following complex operations:

- Large vector unsigned value modular exponentiation
- Large vector unsigned value modular exponentiation using the ‘Chinese Remainders Theorem’ (CRT) method with pre-calculated Q inverse vector
- Modular inversion: Given A and M, calculate B such that $((A \times B) \text{ MOD } M) = 1$
- ECC point addition/doubling on elliptic curve $y^2 = x^3 + ax + b \pmod{p}$ with prime number p and input values a and b to the operation. Adding two identical points automatically performs point doubling.
- ECC point multiplication on elliptic curve $y^2 = x^3 + ax + b \pmod{p}$ with prime number p and input values a and b to the operation. A version of the ‘Montgomery ladder’ algorithm is used to provide side channel attack resistance.

The PKA also contains hardware logic to automatically zero out the PKA RAM buffer to clear out any information that is considered sensitive or secure.

PKA Functional Description

The following sections provide details on the function of the PKA module.

ADSP-2156x PKA Register List

The Public Key Accelerator module (PKA) provides security-related features. A set of registers governs PKA operations. For more information on PKA functionality, see the PKA register descriptions.

Table 34-1: ADSP-2156x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Definitions

The following definitions are helpful when using the PKA module.

Elliptic Curve Cryptography (ECC)

A form of public key cryptography based on elliptic curves over finite fields.

RSA

An acronym for Ron Rivest, Adi Shamir, and Leonard Adleman. It is another form of a public key cryptosystem.

Chinese Remainder Theorem (CRT)

A mathematical theorem used for simplifying time-consuming arithmetic used in public key algorithm computations.

Addition Chaining Table (ACT)

A method of speeding up exponentiation by repeatedly squaring the input and storing the result and reusing the result as input. ACT2 uses a table with 2 address bits (4 entries) and ACT4 uses a table with 4 address bits (16 entries).

PKA Architectural Concepts

The following sections describe the PKA architecture.

Public Key Co-Processor (PKCP)

The Public Key Co-Processor (PKCP) handles the basic large vector processing such as addition, subtraction, multiplication, etc.

Sequencer

The sequencer is small processor that is part of the PKA which handles the more complicated vector processing for public key algorithms. Algorithms include modular exponentiation and the ECC addition and ECC multiply used in Elliptic Curve Cipher algorithms. It executes instructions stored from an internal pre-programmed ROM that handles these operations.

RAM

Input and output vectors are stored in a 4 kB RAM buffer that is part of the MMR space. The address of PKA_RAM is the beginning of the RAM space. This memory is also used as a scratchpad or workspace for the sequencer and PKCP. Programs must place the vectors appropriately following the constraints described in the *Functional Description* section of this chapter.

PKA Block Diagram

The *PKA Block Diagram* shows the top-level block diagram of the PKA engine. The PKA engine is comprised of five parts:

1. Registers for input, output, status, and control
2. Public Key Co-processor (PKCP) module which performs the basic suite of big number (vector) operations typically found in public key cryptography applications
3. Sequencer which controls modular exponentiation, elliptic curve cryptography, and modular inversion operations.
4. Program ROM associated with the PKA engine exclusively for the sequencer
5. PKA RAM holds the large input and output values as well as the workspace/scratchpad required from the sequencer and PKCP for operations.

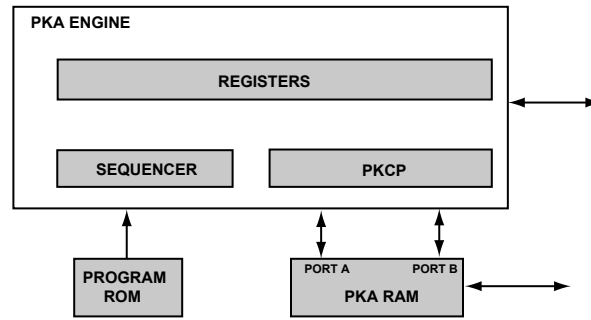


Figure 34-1: PKA Block Diagram

PKCP Vector Operations

The *Summary of PKCP Vector Operations* table lists the arguments and results for each PKCP vector operation.

Table 34-2: Summary of PKCP Vector Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
Multiply	$A \times B \rightarrow C$	Multiplicand	Multiplier	Product	N/A
Add	$A + B \rightarrow C$	Addend	Addend	Sum	N/A
Subtract	$A - B \rightarrow C$	Minuend	Subtracthend	Difference	N/A
AddSub	$A + C - B \rightarrow D$	Addend	Subtracthend	Addend	Result
Right Shift	$A \gg \text{Shift} \rightarrow C$	Input	N/A	Result	N/A
Left Shift	$A \ll \text{Shift} \rightarrow C$	Input	N/A	Result	N/A
Divide	$A \text{ mod } B \rightarrow C,$ $A \text{ div } B \rightarrow D$	Dividend	Divisor	Remainder	Quotient
Modulo	$A \text{ mod } B \rightarrow C$	Dividend	Divisor	Remainder	N/A
Compare	$A = B, A < B, A > B$	Input 1	Input 2	N/A	N/A
Copy	$A \rightarrow C$	Input	N/A	Result	N/A

To obtain correct result, the input vectors must meet the requirements presented in the *Operational Restrictions on Input Vectors for PKCP Operations* table.

Note the following:

- The PKCP does not check input restrictions
- A_Len and B_Len indicate the size of vectors A and B in (32-bit) words
- Max_Len equals 128 (32-bit) words, for example, the standard maximum vector size is 4096 bit

Table 34-3: Operational Restrictions on Input Vectors for PKCP Operations

Function	Requirement
Multiply	$0 < A_Len, B_Len \leq Max_Len$
Add	$0 < A_Len, B_Len \leq Max_Len$
Subtract	$0 < A_Len, B_Len \leq Max_Len$ Result must be positive ($A > B$)
AddSub	$0 < A_Len \leq Max_Len$ (B and C operands have A_Len as length, B_Len ignored) Result must be positive ($(A + C) > B$)
Right Shift	$0 < A_Len \leq Max_Len$
Left Shift	$0 < A_Len \leq Max_Len$
Divide, Modulo	$1 < B_Len \leq A_Len \leq Max_Len$ Most significant 32-bit word of B operand cannot be zero
Compare	$0 < A_Len \leq Max_Len$ (B operand has A_Len as length, B_Len ignored)
Copy	$0 < A_Len \leq Max_Len$

The host processor is responsible for allocating a block of contiguous memory in PKA RAM for the result vectors. The *PKCP Result Vector Memory Allocation* table indicates how much memory is allocated for the result vectors.

Table 34-4: PKCP Result Vector Memory Allocation

Function	Result Vector	Result Vector Length (in 32-bit words)
Multiply	C	$A_Len + B_Len + 6$ (the 6 'scratchpad' words should be discarded)
Add	C	$Max(A_Len, B_Len) + 1$
Subtract	C	$Max(A_Len, B_Len)$
AddSub	D	$A_Len + 1$
Right Shift	C	A_Len
Left Shift	C	$A_Len + 1$ (when Shift Value is non-zero) A_Len (when Shift Value is zero)
Divide	C	Remainder $\rightarrow B_Len + 1$ (one 'scratchpad' word should be discarded)
	D	Quotient $\rightarrow A_Len - B_Len + 1$
Modulo	C	Remainder $\rightarrow B_Len + 1$ (one 'scratchpad' word should be discarded)
Compare	None	Compare updates the PKA_COMPARE register
Copy	C	A_Len

Input vectors for an operation are always allowed to overlap in memory (partially or completely). The *PKCP Result Vector/Input Overlap Restrictions* table gives restrictions for the overlap of output and input vectors of the operations.

Table 34-5: PKCP Result Vector/Input Overlap Restrictions

Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed
Add, Subtract	C	May overlap with A and/or B vector, provided the start address of the C vector does not lie above the start address of the vectors with which it overlaps
AddSub	D	May overlap with A, B and/or C vector, provided the start address of the D vector does not lie above the start address of the vectors with which it overlaps
Right Shift, Left Shift	C	May overlap with A vector, provided the start address of the C vector does not lie above the start address of the A vector
Divide	C	No overlap with A, B, or D vectors allowed
	D	No overlap with A, B, or C vectors allowed
Modulo	C	No overlap with A or B vectors allowed
Compare	None	Compare does not write a result vector
Copy	None	Same restrictions as for right or left shift, copy of a vector to a lower address is always allowed even if source and destination overlap†

†The copy operation can be used to fill memory by breaking the overlap restrictions, but it requires setting up TWO initial (32-bit) words. To zero a block of memory, set the A vector pointer to the block start, set the C vector pointer two words higher and the A vector length to the block length minus two (words). Fill the first two words of the block with constant zero and perform a PKCP copy operation to zero the remainder of the block.

Modular Exponentiation Operations

The *Summary of ExpMod Operations* table summarizes the modular exponentiation operations that the PKA supports.

Table 34-6: Summary of ExpMod Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$C^A \bmod B \rightarrow D$	Exponent, length = A_Len	Modulus, length = B_Len	Base, length = B_Len	Result and Workspace
ExpMod-CRT	See below	Exp P followed by Exp Q at next higher even word address†, both A_Len long	Mod P + buffer word followed by Mod Q at next higher even word address‡, both B_Len long	Q inverse, length = B_Len	Input, Result (both 2xB_Len long) and Workspace

† If A_Len is even, Exp Q follows Exp P immediately – if A_Len is odd, there is one empty word between Exp Q and Exp P.

‡ If B_Len is even, there are two empty words between Mod P and Mod Q – if B_Len is odd, there is one empty (buffer) word between Mod Q and Mod P. Note that the engine may zero the words following Mod P and Mod Q.

The ExpMod-CRT operation performs the following computation steps. (These steps implement Garner’s recombination algorithm after the basic exponentiations.)

- $X \leftarrow (\text{Input mod Mod P}) \text{Exp P mod Mod P}$
- $Y \leftarrow (\text{Input mod Mod Q}) \text{Exp Q mod Mod Q}$
- $Z \leftarrow (((X - Y) \text{ mod Mod P}) \cdot Q \text{ inverse}) \text{ mod Mod P} \cdot \text{Mod Q}$
- $\text{Result} \leftarrow Y + Z$

The ExpMod-ACT2, -ACT4, and -variable functions implement the same mathematical operation but with a differently sized table with pre-calculated *odd powers*. The ExpMod-ACT2 function uses a table with two entries whereas ExpMod-ACT4 uses a table with eight entries. The ACT4 version gives better performance but needs more memory. ExpMod-variable and ExpMod-CRT operations allow the selection of a variable number (from 1 up to and including 16) of odd powers through the register normally used to specify the number of bits to shift for shift operations.

The exponentiation functions are extensions of the set of PKA functions. Input and result vectors are passed the same way as basic PKCP operations. The *Restrictions on Input Vectors for ExpMod Operations* table shows the restrictions on the input and result vectors for the exponentiation operations.

Table 34-7: Restrictions on Input Vectors for ExpMod Operations

Function	Requirements
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$0 < A_Len \leq \text{Max_Len}$ $1 < B_Len \leq \text{Max_Len}$ Modulus B must be odd (for example, the least significant bit must be ONE) $\text{Modulus B} > 2^{32}$ Base C < Modulus B Vectors B and C must be followed by an empty 32-bit <i>buffer</i> word
ExpMod-CRT	$0 < A_Len \leq \text{Max_Len}$ $1 < B_Len \leq \text{Max_Len}$ Mod P and Mod Q must be odd (for example, the least significant bits must be ONE) $\text{Mod P} > \text{Mod Q} > 2^{32}$ (note that Mod P must be larger than Mod Q) Mod P and Mod Q must be co-prime (their GCD must be 1) $0 < \text{Exp P} < (\text{Mod P} - 1)$ $0 < \text{Exp Q} < (\text{Mod Q} - 1)$ $(Q \text{ inverse} \cdot \text{Mod Q}) \equiv 1 \pmod{\text{Mod P}}$ Input < (Mod P · Mod Q) Mod P and Mod Q must be followed by an empty 32-bit <i>buffer</i> word

The *ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR* table shows the required scratchpad sizes for the exponentiation operations. These sizes depend on the PKA type. The ‘M_Len’ used in the table is the ‘real’ Modulus length in 32-bit words, for example, without trailing zero words at the end. (This description also applies to Mod P in an ExpMod-CRT operation and Modulus B in the other operations.) If the last word of the modulus vector as given is non-zero, ‘M_Len’ equals B_Len.

Table 34-8: ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR

Function	Scratchpad Area Size (in 32-bit words), Result Vector is either M_Len or 2xM_Len 32-bit words long
ExpMod-ACT2	5 x (M_Len + 2)
ExpMod-ACT4	11 x (M_Len + 2)
ExpMod-variable	(# odd powers + 3) x (M_Len + 2)
ExpMod-CRT	(# odd powers + 3) x (M_Len + 2) + (M_Len + 2 – (M_Len MOD 2))

NOTE: During execution of an ExpMod-ACT2, -ACT4 or -variable operation, the last 34 bytes of the PKA RAM are used as the general scratchpad for the sequencer program execution. The ExpMod-CRT operation requires the last 72 bytes of the PKA RAM as the scratchpad. These (fixed location) areas may not overlap with any of the input vectors and/or the D vector scratchpad area. They can be used freely when executing basic PKCP operations.

Table 34-9: ExpMod Scratchpad Area / Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	D	Scratchpad area starting at D may not overlap with any of the other vectors, except that base C may be co-located with result vector D to save space (for example, PKA_CPTR = PKA_DPTR is allowed).
ExpMod-CRT	D	Scratchpad area starting at D may not overlap with any of the other vectors. This area is also the location of the main input vector (with length 2 x B_Len)

The *Maximum Number of Odd Powers* table indicates the maximum number of odd powers that can be used for different standard PKA RAM sizes and PKA types (non-CRT operations using PKA_CPTR = PKA_DPTR). As a rule of thumb, for optimal performance, use one odd power for *Verify* operations and 4 (or as many as the implemented PKA RAM size allows) for *Sign* operations. Note the following points about odd powers:

- Using more than eight odd powers is not advisable as the speed advantage for each extra odd power decreases rapidly (and can even become negative for short exponent vector lengths due to the extra pre-processing required).
- The maximum number of odd powers is 16 (limited by the firmware). All ‘16 odd powers’ entries in the table above hit this limit – they are not limited by the PKA RAM size.

Table 34-10: Maximum Number of Odd Powers

Operation	Modulus and Exponent Sizes	Maximum Number of Odd Powers
Non-CRT	1024 bits	16
	2048 bits	10
	4096 bits	2
CRT	2 × 512 bits	16
	2 × 1024 bits	16
	2 × 2048 bits	6

The *2K-bit Modular Exponentiation PKA RAM Allocation Examples* table shows example PKA RAM vector allocations for modular exponentiation operations with and without using CRT. The *free space* start address is the first free byte following the vector workspace. The sequencer execution scratchpad of 34 bytes (non-CRT) or 72 bytes (using CRT) must fit between this address and the end of the PKA RAM. Note that the non-CRT operations use `PKA_CPTR = PKA_DPTR` to save space.

Table 34-11: 2K-bit Modular Exponentiation PKA RAM Allocation Examples

Operation	(sub-)vector	Start address (Byte Offset)	Size (words)	Buffer (words)
non-CRT (<code>PKA_ALEN = 0x040</code> , <code>PKA_BLEN = 0x040</code> , 4 odd-powers)	Exponent	0x000 (<code>PKA_APTR = 0x000</code>)	64	0
	Modulus	0x100 (<code>PKA_BPTR = 0x040</code>)	64	2
	Base	0x208 (<code>PKA_CPTR = 0x082</code>)	64	2
	Result	0x208 (<code>PKA_DPTR = 0x082</code>)	64	2
	Vector Workspace	0x208 (= Result)	7 × (64+2)=462	0
	Free space	0x940 (2368 bytes used)	-	-
using CRT (<code>PKA_ALEN = 0x020</code> , <code>PKA_BLEN = 0x020</code> , 4 odd-powers)	Exp P	0x000 (<code>PKA_APTR = 0x000</code>)	32	0
	Exp Q	0x080	32	0
	Mod P	0x100 (<code>PKA_BPTR = 0x040</code>)	32	2
	Mod Q	0x188	32	2
	Q inverse	0x210 (<code>PKA_CPTR = 0x084</code>)	32	0
	Input, Result	0x290 (<code>PKA_DPTR = 0x0A4</code>)	64	0
	Vector workspace	0x290 (= Result)	7 × (32+2)+32+2-0 = 272	0
	Free space	0x6D0 (1744 bytes used)	-	-

The following example in pseudo-code describes the execution of a non-CRT modular exponentiation operation using a 512 bits modulus and a 160 bits exponent, using actual test vectors:

```
// Perform a 512/160 bits
modular exponentiation without CRT (using 4 'odd-powers')
// Exponent equals value 0x8FD84098_8A0930CC_9CDC1E8A_B246EB46_2D39F064
// write as vector A to PKA
```



```

RAM Byte offset 0x000:
Write PKA_RAM_BASE+0x000+0x00
0x2D39F064
Write PKA_RAM_BASE+0x000+0x04
0xB246EB46
Write PKA_RAM_BASE+0x000+0x08
0x9CDC1E8A
Write PKA_RAM_BASE+0x000+0x0C
0x8A0930CC
Write PKA_RAM_BASE+0x000+0x10
0x8FD84098
// Modulus equals value 0xF42F559D
1877CA5F_449492B9_42DC7C01_...
// A3C9085B_7236A085_2102B000_A093C6B4_...
// 9D0EDA0C_292DE841_29C23723_4048BDA3_...
// 373C4C9F_45CF15A7_5F049ABF_D8A01B9B
// write as vector B to PKA
RAM Byte offset 0x018 (following exp at next aligned 64-bit word):
Write PKA_RAM_BASE+0x018+0x00
0xD8A01B9B
Write PKA_RAM_BASE+0x018+0x04
0x5F049ABF
Write PKA_RAM_BASE+0x018+0x08
0x45CF15A7
Write PKA_RAM_BASE+0x018+0x0C
0x373C4C9F
Write PKA_RAM_BASE+0x018+0x10
0x4048BDA3
Write PKA_RAM_BASE+0x018+0x14
0x29C23723
Write PKA_RAM_BASE+0x018+0x18
0x292DE841
Write PKA_RAM_BASE+0x018+0x1C
0x9D0EDA0C
Write PKA_RAM_BASE+0x018+0x20
0xA093C6B4
Write PKA_RAM_BASE+0x018+0x24
0x2102B000
Write PKA_RAM_BASE+0x018+0x28
0x7236A085
Write PKA_RAM_BASE+0x018+0x2C
0xA3C9085B
Write PKA_RAM_BASE+0x018+0x30
0x42DC7C01
Write PKA_RAM_BASE+0x018+0x34
0x449492B9
Write PKA_RAM_BASE+0x018+0x38
0x1877CA5F
Write PKA_RAM_BASE+0x018+0x3C
0xF42F559D

```

```

// Base equals value 0x3D291F48_49064887_1149594B_67935110_...
// 14EB8FF0_AB291F3A_54A1B4D1_5E611E44_...
// C989251B_44904B45_0B060482_317F8352_...
// 18CE440E_9BF509F1_6EAF26F2_95F19F12
// write as vector C to PKA
RAM Byte offset 0x060 (following mod after buffer + align words):
Write PKA_RAM_BASE+0x060+0x00
0x95F19F12
Write PKA_RAM_BASE+0x060+0x04
0x6EAF26F2
Write PKA_RAM_BASE+0x060+0x08
0x9BF509F1
Write PKA_RAM_BASE+0x060+0x0C
0x18CE440E
Write PKA_RAM_BASE+0x060+0x10
0x317F8352
Write PKA_RAM_BASE+0x060+0x14
0x0B060482
Write PKA_RAM_BASE+0x060+0x18
0x44904B45
Write PKA_RAM_BASE+0x060+0x1C
0xC989251B
Write PKA_RAM_BASE+0x060+0x20
0x5E611E44
Write PKA_RAM_BASE+0x060+0x24
0x54A1B4D1
Write PKA_RAM_BASE+0x060+0x28
0xAB291F3A
Write PKA_RAM_BASE+0x060+0x2C
0x14EB8FF0
Write PKA_RAM_BASE+0x060+0x30
0x67935110
Write PKA_RAM_BASE+0x060+0x34
0x1149594B
Write PKA_RAM_BASE+0x060+0x38
0x49064887
Write PKA_RAM_BASE+0x060+0x3C
0x3D291F48
// The result value and scratchpad
(vector D) may be co-located with the base vector C for
// a normal modular exponentiation,
so these are located at PKA RAM Byte offset 0x060 too.
// Load pointer and length
registers:
Write PKA_APTR 0x000>>2 //
Exponent pointer
Write PKA_BPTR 0x018>>2 //
Modulus pointer
Write PKA_CPTR 0x060>>2 //
Base pointer

```

```

Write PKA_DPTR 0x060>>2 //
Result/scratchpad pointer
Write PKA_ALENGTH 0x00000005
// Exponent length in 32-bit words
Write PKA_BLENGTH 0x00000010
// Mod/base/result length in 32-bit words
// Start modular exponentiation
and wait until it's done:
Write PKA_SHIFT 0x00000004
// Number of 'odd powers'
Write PKA_FUNCTION 0x0000E000
// 'Run' bit set, 'Sequencer Operations' = 0b110
Wait PKA_FUNCTION[15] == '0'
// 'Run' bit clears itself - Host can also use interrupt!
// Result value equals 0xA497BF8B_DB729088_954005B0_B5CA6691_...
// A3EC491B_091A3D62_03C24214_0863A389_...
// 0C7C03CD_2333E231_35EC10ED_8F91281C_...
// 30F4253B_FE38FAFB_BB4A39DB_C14F2661
// written as vector D at
PKA RAM Byte offset 0x060:
Check PKA_RAM_BASE+0x060+0x00
== 0xC14F2661
Check PKA_RAM_BASE+0x060+0x04
== 0xBB4A39DB
Check PKA_RAM_BASE+0x060+0x08
== 0xFE38FAFB
Check PKA_RAM_BASE+0x060+0x0C
== 0x30F4253B
Check PKA_RAM_BASE+0x060+0x10
== 0x8F91281C
Check PKA_RAM_BASE+0x060+0x14
== 0x35EC10ED
Check PKA_RAM_BASE+0x060+0x18
== 0x2333E231
Check PKA_RAM_BASE+0x060+0x1C
== 0x0C7C03CD
Check PKA_RAM_BASE+0x060+0x20
== 0x0863A389
Check PKA_RAM_BASE+0x060+0x24
== 0x03C24214
Check PKA_RAM_BASE+0x060+0x28
== 0x091A3D62
Check PKA_RAM_BASE+0x060+0x2C
== 0xA3EC491B
Check PKA_RAM_BASE+0x060+0x30
== 0xB5CA6691
Check PKA_RAM_BASE+0x060+0x34
== 0x954005B0
Check PKA_RAM_BASE+0x060+0x38
== 0xDB729088

```

```
Check PKA_RAM_BASE+0x060+0x3C
== 0xA497BF8B
```

Modular Inversion

Besides modular exponentiation, the sequencer also controls modular inversion operations.

Table 34-12: Summary of ModInv Operation

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ModInv	$A^{-1} \bmod B \rightarrow D$	NumToInvert, length = A_Len	Modulus, length = B_Len	Not Used	Result and Workspace

The above function appears to be an extension of the set of basic PKCP functions with the following exceptions:

- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register field is used to return info on the operation's result.

Table 34-13: PKA_SHIFT Result Values for ModInv Operation

Function	PKA_SHIFT Register Field Value At Conclusion
ModInv	0 → success; VectorD holds result 7 → no inverse exists ($\text{GCD}(A, B) \neq 1$, for example, A and B have common factors); result undefined 31 → error, modulus even; result undefined other values are reserved

The following tables list the restrictions on the input and result vectors for the ModInv operation:

Table 34-14: Operational Restrictions on Input Vectors for the ModInv Operation

Function	Requirements
ModInv	$0 < A_Len \leq \text{Max_Len}$ $0 < B_Len \leq \text{Max_Len}$ Modulus B must be odd (for example, the least significant bit must be ONE) Modulus B may not have value 1 (result is undefined, no error indicated) The highest word of the modulus vector, as indicated by B_Len, may not be zero.

Table 34-15: ModInv Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ModInv	D	Scratchpad area starting at D may not overlap with any of the other vectors

The following table shows the required scratchpad sizes for the ModInv Operation:

Table 34-16: ModInv Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	Scratchpad area size (in 32-bit words), Result Vector is B_Length 32-bit words long
ModInv	$5 \times (M + \varepsilon(M))$, with $M = \text{Max}(A_Length, B_Length)$ $\varepsilon(n) = 2 + (n \text{ MOD } 2)$, for example, 2 (for n even) or 3 (for n odd)

NOTE: During execution of a ModInv operation, the last 34 bytes of the PKA RAM are used as general scratchpad for the sequencer program execution. This (fixed location) area may not overlap with any of the input vectors and/or the D vector scratchpad area during execution.

Modular Inversion with an Even Modulus

The ModInv operation requires the modulus to be odd. At first, this requirement appears to make the operation useless in the case of RSA key generation where the private key exponent d is derived from a chosen public exponent e as follows:

$$d = \text{ModInv}(e, \varphi); \text{ where } \varphi = (p-1) \times (q-1) \text{ and } p \text{ and } q \text{ both prime}$$

Note that φ is even. However, since e must be odd (otherwise no inverse exists), d can be calculated as:

$$d = (1 + (\varphi \times (e - \text{ModInv}(\varphi, e)))) / e$$

With four more basic PKCP operations, ModInv can also be used to find inverse values in case the modulus is even.

Modular Inversion with a Prime Modulus

Modular inversion can be performed with a modular exponentiation using the modulus value minus two as exponent, provided that the modulus value is a prime. This is due to the following:

$$(A^M) \text{ mod } M = A \Rightarrow$$

$$(A^{M-1}) \text{ mod } M = 1 \Rightarrow$$

$$(A^{M-2}) \text{ mod } M = A^{-1} \text{ (mod } M)$$

Under the constraint that M is a prime value.

Especially with the large PKA engines containing an LNME, it is worthwhile to check whether this method is faster than using the ModInv operation directly. The modulus values for the ECC curves supported by this PKA engine must be prime, so this method can be used in ECDSA operations.

ECC Operations

Besides modular exponentiation and modular inversion, the sequencer also controls ECC operations.

Table 34-17: Summary of ECC Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ECC-ADD	Point addition/ doubling† on elliptic curve: $y^2 = x^3 + ax + b \pmod{p}$ $\text{pntA} + \text{pntC} \rightarrow \text{pntD}$	pntA.x followed‡ by pntA.y both B_Len long (A_Len <i>not</i> used)	Curve parameter p followed‡ by a (b is not needed) all B_Len long	pntC.x followed by pntC.y both B_Len long	Result, for example, pntD.x followed‡ by pntD.y and workspace
ECC-MUL	Point multiplication on elliptic curve: $y^2 = x^3 + ax + b \pmod{p}$ $k \times \text{pntC} \rightarrow \text{pntD}$	Scalar k A_Len long	Curve parameter p followed‡ by a and b all B_Len long.	pntC.x followed‡ by pntC.y both B_Len long	Result, for example pntD.x followed‡ by pntD.y and workspace

† If $\text{pntA} = \text{pntC}$, a point doubling operation is performed automatically.

‡ All input components must be located on a 64-bit boundary and must have extra 'buffer' words (of 32 bits each) after their most significant word. ϵ must be 3 (B_Len odd) or 2 (B_Len even). Each result component (for example, pntD.x , pntD.y) is followed by ϵ buffer (zero) words.

The above functions appear to be extensions of the set of PKCP basic functions with the following exceptions:

- Input and result vectors can now be composite (for example, consist of two or three equal-sized subvectors)
- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register is used to return info on the result of the operation

Table 34-18: PKA_SHIFT Result Values for ECC Operations

Function	PKA_SHIFT Register Field Value at Conclusion
ECC-ADD	0 → success; VectorD holds result point
ECC-MUL	7 → result is point-at-infinity; VectorD result point undefined
	31 → error, (p not odd, p too short, etc); VectorD result point undefined
	Other values are reserved

The following tables below list the restrictions on the input and result vectors for the ECC operations.

Table 34-19: Operational Restrictions on Input Vectors for ECC Operations

Function	Requirements
ECC-ADD	<p>$1 < B_Len \leq 24$ (maximum vector length is 768 bits)</p> <p>Modulus p must be a prime $> 2^{63}$</p> <p>Effective modulus size (in bits) must be a multiple of 32†</p> <p>The highest word of the modulus vector, as indicated by B_Len, may not be zero.</p> <p>$a < p$ and $b < p$</p> <p>$pntA$ and $pntC$ must be on the curve (this condition is not checked)</p> <p>Neither $pntA$ nor $pntC$ can be the point-at-infinity, although ECC-ADD can return this point as a result</p>
ECC-MUL	<p>$0 < A_Len \leq 24$ (maximum vector length is 768 bits)</p> <p>$1 < B_Len \leq 24$ (maximum vector length is 768 bits)</p> <p>Modulus p must be a prime $> 2^{63}$</p> <p>Effective modulus size (in bits) must be a multiple of 32†</p> <p>The highest word of the modulus vector, as indicated by B_Len, may not be zero.</p> <p>$a < p$ and $b < p$</p> <p>$pntC$ must be on the curve (this condition is not checked)</p> <p>$pntC$ cannot be the point-at-infinity, although ECC-MUL can return this point as a result</p>

† Modulus lengths of 112 and 521 bits are exceptions to this rule.

Table 34-20: ECC Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ECC-ADD ECC-MUL	D	Scratchpad area starting at D may not overlap with any of the other vectors

The *>ECC Result Vector/Scratchpad Area Memory Allocation* table shows the required scratchpad sizes for the ECC operations:

Table 34-21: ECC Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	Scratchpad area size (in 32-bit words), Result Vector is $2 \times (B_Length + \epsilon(B_Length))$ 32-bit words long
ECC-ADD	$2 \times L + 5 \times M$, where $L = B_Length + \epsilon(B_Length)$ $M = B_Length + 1 + \epsilon(B_Length + 1)$
ECC-MUL	$18 \times L + \text{Max}(8, L)$, where $L = (B_Length + \epsilon(B_Length))$

† $\epsilon(n) = 2 + (n \text{ MOD } 2)$, for example, 2 (for n even) or 3 (for n odd)

NOTE: During execution of an ECC-ADD or ECC-MUL operation, the last 72 bytes of the PKA RAM are used as a general scratchpad for the sequencer program execution. These (fixed location) areas must not overlap with any of the input vectors and the D vector scratchpad area during execution.

The *ECC Point Multiplication PKA RAM Allocation Examples* table shows example PKA RAM vector allocations for ECC point multiplication operations. The *free space* start address is the first free byte following the vector scratchpad. The sequencer execution scratchpad of 72 bytes must fit between this address and the end of the PKA RAM. Because of this requirement, a 521-bit ECC point multiplication cannot be performed with 2K byte PKA RAM.

Table 34-22: ECC Point Multiplication PKA RAM Allocation Examples

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
192 bits (=6 words, PKA_ALEN=0x006, PKA_BLEN=0x006)	Scalar k	0x000 (PKA_APTR = 0x000)	6	0
	p	0x018 (PKA_BPTR = 0x006)	6	2
	a	0x038	6	2
	b	0x058	6	2
	PntC.x (base)	0x078 (PKA_CPTR = 0x01E)	6	2
	PntC.y (base)	0x098	6	2
	PntD.x (result)	0x0B8 (PKA_DPTR = 0x02E)	6	2
	PntD.y (result)	0x0D8	6	0
	Vector scratchpad	0x0B8 (= PntD.x)	(18 × 8) + 8 = 152	0
	Free space	0x318 (792 bytes used)	-	-
384 bits (=12 words, ALENGTH=0x00C, BLENGTH=0x00C)	Scalar k	0x000 (PKA_APTR = 0x000)	12	0
	p	0x030 (PKA_BPTR = 0x00C)	12	2
	a	0x068	12	2
	b	0x0A0	12	2
	PntC.x (base)	0x0D8 (PKA_CPTR = 0x036)	12	2
	PntC.y (base)	0x110	12	2
	PntD.x (result)	0x148 (PKA_DPTR = 0x052)	12	2
	PntD.y (result)	0x180	12	0
	Vector scratchpad	0x148 (= PntD.x)	(18 × 14) + 14 = 266	0
	Free space	0x570 (1392 bytes used)	-	-
521 bits (=17 words, ALENGTH=0x011, BLENGTH=0x011)	Scalar k	0x000 (PKA_APTR = 0x000)	17	1 (to align p)
	p	0x048 (PKA_BPTR = 0x012)	17	3
	a	0x098	17	3
	b	0x0E8	17	3
	PntC.x (base)	0x138 (PKA_CPTR = 0x04E)	17	3
	PntC.y (base)	0x188	17	3
	PntD.x (result)	0x1D8 (PKA_DPTR = 0x076)	17	3

Table 34-22: ECC Point Multiplication PKA RAM Allocation Examples (Continued)

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
	PntD.y (result)	0x228	17	0
	Vector scratchpad	0x1D8 (= PntD.x)	$(18 \times 20) + 20 = 380$	0
	Free space	0x7C8 (1992 bytes used)	-	-

The following example in pseudo-code describes the execution of a 192 bits ECC point multiplication, using actual test vectors (the curve parameters and generator point are from standard curve 'secp192r1').

```
// Perform a 192 bits ECC point multiplication using PKA RAM layout from table
// above.
// Scalar 'k' equals value 0x8D98D058_9EFD018A_C9BCF3CF_2C33AEC0_24867D7F_6ADACBFF
// write as vector A to PKA RAM Byte offset 0x000:

    Write PKA_RAM_BASE+0x000+0x00 0x6ADACBFF
    Write PKA_RAM_BASE+0x000+0x04 0x24867D7F
    Write PKA_RAM_BASE+0x000+0x08 0x2C33AEC0
    Write PKA_RAM_BASE+0x000+0x0C 0xC9BCF3CF
    Write PKA_RAM_BASE+0x000+0x10 0x9EFD018A
    Write PKA_RAM_BASE+0x000+0x14 0x8D98D058
// Curve parameter 'p' equals value
0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFF
// write as 1st part of vector B immediately following vector A at PKA RAM Byte
// offset 0x018
// (no buffer word needed after 'k' vector, 64-bit alignment is OK):
    Write PKA_RAM_BASE+0x018+0x00 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x04 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x08 0xFFFFFFFFE
    Write PKA_RAM_BASE+0x018+0x0C 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x10 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x14 0xFFFFFFFF
// Curve parameter 'a' equals value
0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFC
// write as 2nd part of vector B after one buffer word and one re-alignment word
// at 0x038:
    Write PKA_RAM_BASE+0x038+0x00 0xFFFFFFFFC
    Write PKA_RAM_BASE+0x038+0x04 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x08 0xFFFFFFFFE
    Write PKA_RAM_BASE+0x038+0x0C 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x10 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x14 0xFFFFFFFF
// Curve parameter 'b' equals value
0x64210519_E59C80E7_0FA7E9AB_72243049_FEB8DEEC_C146B9B1
// write as 3rd part of vector B after one buffer word and one re-alignment word
// at 0x058:
    Write PKA_RAM_BASE+0x058+0x00
0xC146B9B1
```

```

Write PKA_RAM_BASE+0x058+0x04 0xFEB8DEEC
Write PKA_RAM_BASE+0x058+0x08 0x72243049
Write PKA_RAM_BASE+0x058+0x0C 0x0FA7E9AB
Write PKA_RAM_BASE+0x058+0x10 0xE59C80E7
Write PKA_RAM_BASE+0x058+0x14 0x64210519
// X-coord of generator point is value
0x188DA80E_B03090F6_7CBF20EB_43A18800_F4FF0AFD_82FF1012
// write as 1st part of vector C following vector B after buffer + alignment
words at 0x078:
Write PKA_RAM_BASE+0x078+0x00 0x82FF1012
Write PKA_RAM_BASE+0x078+0x04 0xF4FF0AFD
Write PKA_RAM_BASE+0x078+0x08 0x43A18800
Write PKA_RAM_BASE+0x078+0x0C 0x7CBF20EB
Write PKA_RAM_BASE+0x078+0x10 0xB03090F6
Write PKA_RAM_BASE+0x078+0x14 0x188DA80E
// Y-coord of generator point is value
0x07192B95_FFC8DA78_631011ED_6B24CDD5_73F977A1_1E794811
// write as 2nd part of vector C after one buffer word and one re-alignment word
at 0x098:
Write PKA_RAM_BASE+0x098+0x00 0x1E794811
Write PKA_RAM_BASE+0x098+0x04 0x73F977A1
Write PKA_RAM_BASE+0x098+0x08 0x6B24CDD5
Write PKA_RAM_BASE+0x098+0x0C 0x631011ED
Write PKA_RAM_BASE+0x098+0x10 0xFFC8DA78
Write PKA_RAM_BASE+0x098+0x14 0x07192B95
// The result point and scratchpad (vector D) follow vector C after one buffer
word and one
// re-alignment word, so these are located at PKA RAM Byte offset 0x0B8.
// Load pointer and length registers:
Write PKA_APTR 0x000>>2 // Scalar 'k' pointer
Write PKA_BPTR 0x018>>2 // Curve parameters 'p', 'a' & 'b' pointer
Write PKA_CPTR 0x078>>2 // Generator point X & Y coordinates pointer
Write PKA_DPTR 0x0B8>>2 // Result point X & Y coordinates/scratchpad pointer
Write PKA_ALENGTH 0x00000006 // Scalar 'k' length in 32-bit words
Write PKA_BLENGTH 0x00000006 // Curve parameters and coordinate lengths in 32-
bit words
// Start ECC point multiplication and wait until it's done:
Write PKA_FUNCTION 0x0000D000
// 'Run' bit set, 'Sequencer Operations' = 0b101
Wait PKA_FUNCTION[15] == '0'
// 'Run' bit clears itself - Host can also use interrupt!
Check PKA_SHIFT == 0x00000000
// Shift field value 0 indicates success - check this
// X-coord of result point is value
0x759B9F39_0E81D268_18C82BB9_CB42BCF5_0E0AE958_85BA3097
// written as 1st part of vector D at PKA RAM Byte offset 0x0B8:
Check PKA_RAM_BASE+0x0B8+0x00== 0x85BA3097
Check PKA_RAM_BASE+0x0B8+0x04== 0x0E0AE958
Check PKA_RAM_BASE+0x0B8+0x08== 0xCB42BCF5
Check PKA_RAM_BASE+0x0B8+0x0C== 0x18C82BB9

```

```

Check PKA_RAM_BASE+0x0B8+0x10== 0x0E81D268
Check PKA_RAM_BASE+0x0B8+0x14== 0x759B9F39
// Y-coord of result point is value
0xECA14640_F92EFF07_CAF2BD55_3FBE28EF_D043F28E_1CC3D238
// written as 2nd part of vector D at PKA RAM Byte offset 0x0D8:
Check PKA_RAM_BASE+0x0D8+0x00== 0x1CC3D238
Check PKA_RAM_BASE+0x0D8+0x04== 0xD043F28E
Check PKA_RAM_BASE+0x0D8+0x08== 0x3FBE28EF
Check PKA_RAM_BASE+0x0D8+0x0C== 0xCAF2BD55
Check PKA_RAM_BASE+0x0D8+0x10== 0xF92EFF07
Check PKA_RAM_BASE+0x0D8+0x14== 0xECA14640

```

ADSP-2156x PKA Register Descriptions

Public Key Accelerator (PKA) contains the following registers.

Table 34-23: ADSP-2156x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Vector_A Length

During execution of basic PKCP operations, the `PKA_ALEN` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_ALEN` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

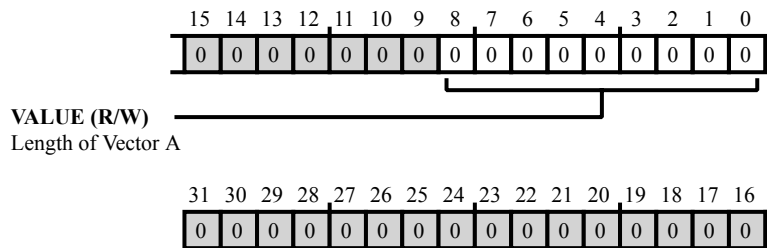


Figure 34-2: `PKA_ALEN` Register Diagram

Table 34-24: `PKA_ALEN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector A. Length (in 32-bit words) of Vector A.

PKA Vector_A Address

During execution of basic PKCP operations, the `PKA_APTR` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_APTR` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

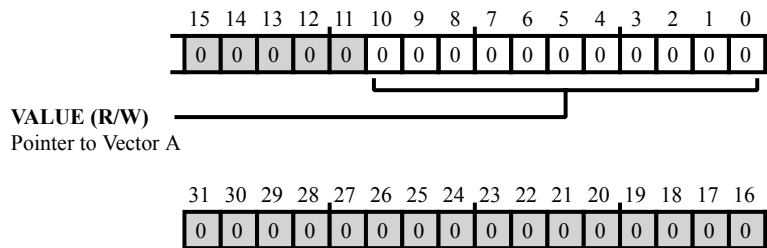


Figure 34-3: `PKA_APTR` Register Diagram

Table 34-25: `PKA_APTR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector A. The <code>PKA_APTR.VALUE</code> bit field is the location of Vector A within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Vector_B Length

During execution of basic PKCP operations, the `PKA_BLEN` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_BLEN` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

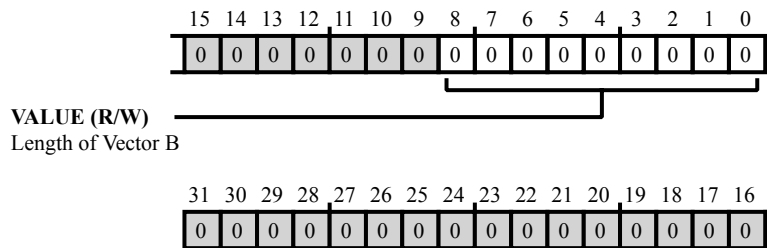


Figure 34-4: PKA_BLEN Register Diagram

Table 34-26: PKA_BLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector B. Length (in 32-bit words) of Vector B.

PKA Vector_B Address

During execution of basic PKCP operations, the `PKA_BPTR` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_BPTR` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

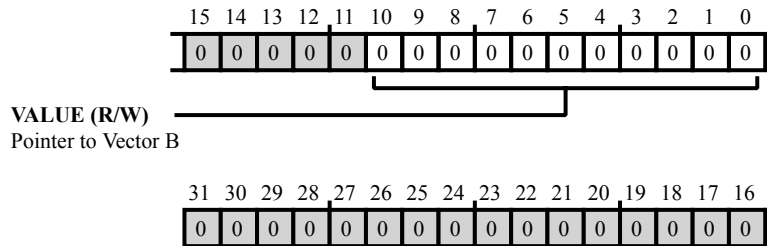


Figure 34-5: `PKA_BPTR` Register Diagram

Table 34-27: `PKA_BPTR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector B. The <code>PKA_BPTR.VALUE</code> bit field is the location of Vector B within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Compare Result

The `PKA_COMPARE` register provides the result of a basic PKCP Compare operation. It is updated when the `PKA_FUNC.RUN` bit is reset at the end of that operation. The status after a complex sequencer operation is unknown.

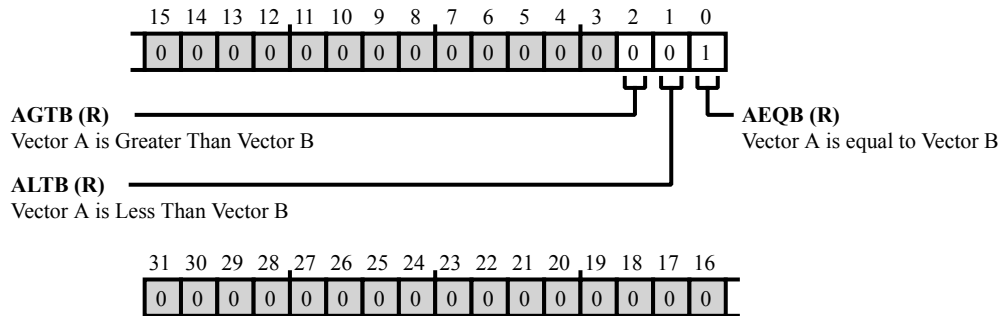


Figure 34-6: PKA_COMPARE Register Diagram

Table 34-28: PKA_COMPARE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	AGTB	Vector A is Greater Than Vector B. The <code>PKA_COMPARE.AGTB</code> bit shows the result of the basic compare operation is PKCP <code>Vector_A</code> is greater than <code>Vector_B</code> .
1 (R/NW)	ALTB	Vector A is Less Than Vector B. The <code>PKA_COMPARE.ALTB</code> bit shows the result of the basic compare operation is <code>Vector_A</code> is less than <code>Vector_B</code> .
0 (R/NW)	AEQB	Vector A is equal to Vector B. The <code>PKA_COMPARE.AEQB</code> bit shows the result of the basic compare operation is <code>Vector_A</code> is equal to <code>Vector_B</code> .

PKA Vector_C Address

During execution of basic PKCP operations, the `PKA_CPTR` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_CPTR` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

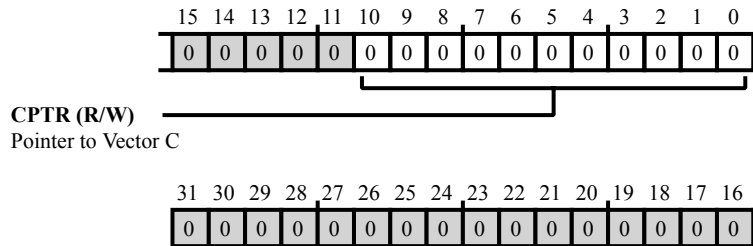


Figure 34-7: PKA_CPTR Register Diagram

Table 34-29: PKA_CPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	CPTR	Pointer to Vector C. The <code>PKA_CPTR.CPTR</code> bit field is the location of Vector C within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Most-Significant-Word of Divide Remainder

The `PKA_DIVMSW` register indicates the (32-bit word) address in the PKA RAM where the most significant non-zero 32-bit word of the Remainder result for the basic Divide and Modulo operations is stored. Bits [4:0] are loaded with the bit number of the most significant non-zero bit in the most significant non-zero word when MS one control bit is set. For Divide, Modulo and MS one reporting, this register is updated when the `PKA_FUNC.RUN` bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating bits [4:0] of this register with the actual result's most significant bit location is done near the end of the operation. Note that the result is only meaningful if no errors were detected and that for ECC operations, the `PKA_DIVMSW` register provides information for the x-coordinate of the result point only.

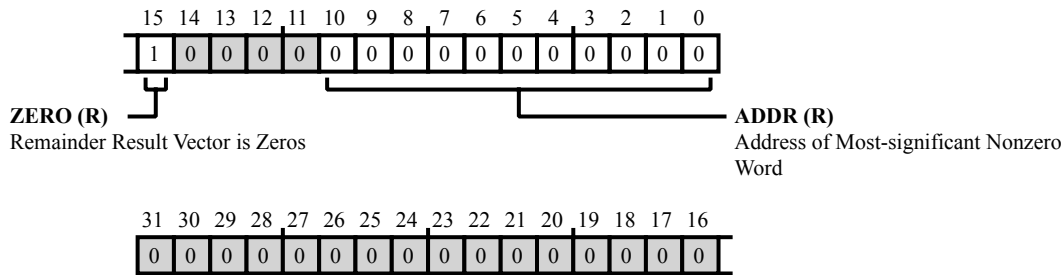


Figure 34-8: `PKA_DIVMSW` Register Diagram

Table 34-30: `PKA_DIVMSW` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Remainder Result Vector is Zeros. The <code>PKA_DIVMSW.ZERO</code> bit shows the remainder result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The <code>PKA_DIVMSW.ADDR</code> bit shows the address of the most significant non-zero 32-bit word of the remainder result vector in PKA RAM.

PKA Vector_D Address

During execution of basic PKCP operations, the `PKA_DPTR` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_DPTR` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

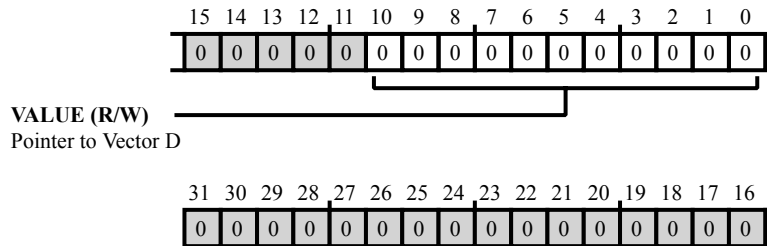


Figure 34-9: PKA_DPTR Register Diagram

Table 34-31: PKA_DPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector D. The <code>PKA_DPTR.VALUE</code> bit field is the location of Vector D within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Function

The `PKA_FUNC` register contains the control bits to start basic PKCP as well as complex sequencer operations. The `PKA_FUNC.RUN` bit can be used to poll for the completion of the operation. Modifying bits [11:0] is made impossible during the execution of a basic PKCP operation.

During the execution of Sequencer controlled complex operations, this register is modified - the `PKA_FUNC.RUN` and `PKA_FUNC.STALLRSLT` bits are set to zero at the conclusion, but other bits are undefined.

Continuously reading this register to poll the `PKA_FUNC.RUN` bit is NOT allowed when executing complex sequencer operations (the sequencer cannot access the PKCP when this is done).

Leave at least one SCLK cycle between poll operations.

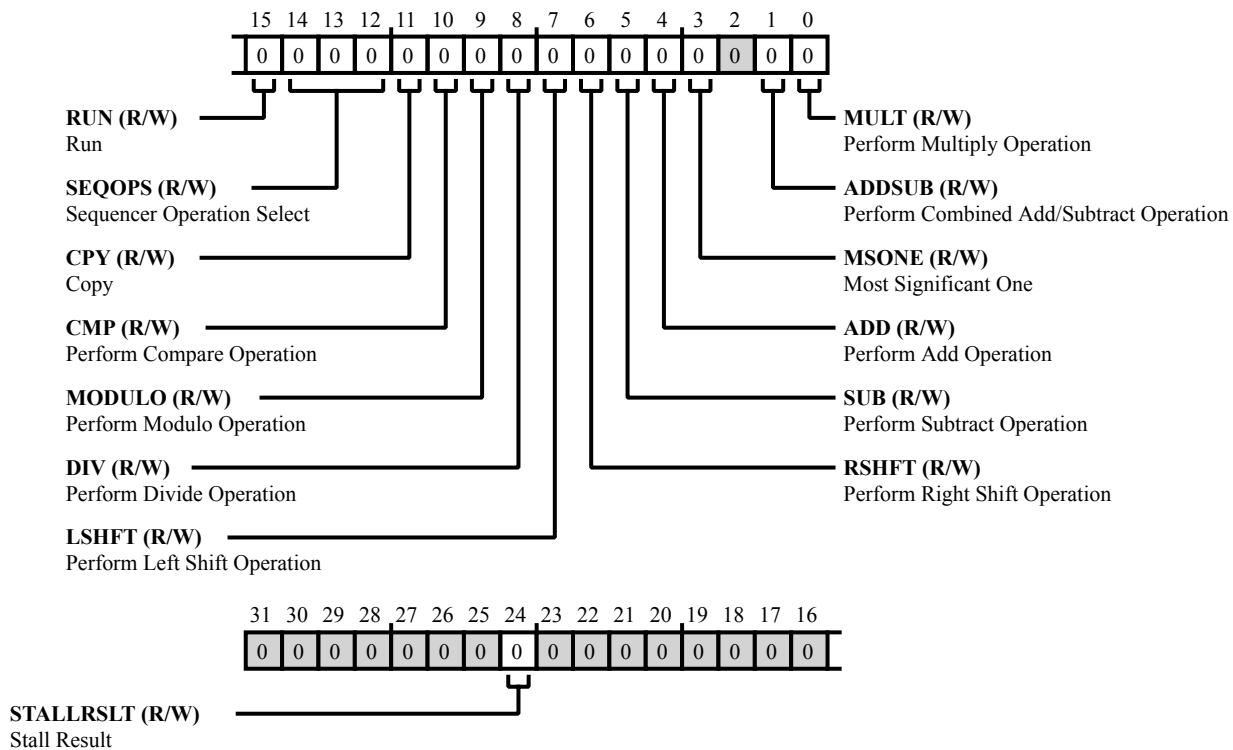


Figure 34-10: `PKA_FUNC` Register Diagram

Table 34-32: PKA_FUNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration																
24 (R/W)	STALLRSLT	<p>Stall Result.</p> <p>When the <code>PKA_FUNC.STALLRSLT</code> bit is set, updating the <code>PKA_COMPARE</code>, <code>PKA_RESULTMSW</code> and <code>PKA_DIVMSW</code> registers, as well as resetting the Run bit, is stalled beyond the point that a running operation is actually finished. Use this to allow software enough time to read results from a previous operation when the newly started operation is known to take only a short amount of time. If a result is waiting, the result registers is updated and the Run bit is reset in the clock cycle following writing the Stall Result bit back to 0. The Stall result function may only be used for basic PKCP operations.</p>																
15 (R/W)	RUN	<p>Run.</p> <p>Set the <code>PKA_FUNC.RUN</code> bit to instruct the PKA module to begin processing the basic PKCP or complex Sequencer operation. This bit is reset low automatically when the operation is complete. The complement of this bit is output as the <code>pkaint1</code> interrupt.</p> <p>After a reset, the Run bit is always set to 1b but the first Sequencer firmware instruction sets this bit to 0 immediately after the hardware reset is released. A few clock cycles are needed before the first instruction is executed and the Run bit state has been propagated.</p>																
14:12 (R/W)	SEQOPS	<p>Sequencer Operation Select.</p> <p>The <code>PKA_FUNC.SEQOPS</code> bit field select the complex Sequencer operation to perform.</p> <table border="1"> <tbody> <tr> <td>0</td> <td>None</td> </tr> <tr> <td>1</td> <td>ExpMod-CRT</td> </tr> <tr> <td>2</td> <td>ExpMod-ACT4</td> </tr> <tr> <td>3</td> <td>ECC-ADD</td> </tr> <tr> <td>4</td> <td>ExpMod-ACT2</td> </tr> <tr> <td>5</td> <td>ECC-MUL</td> </tr> <tr> <td>6</td> <td>ExpMod-variable</td> </tr> <tr> <td>7</td> <td>ModInv</td> </tr> </tbody> </table>	0	None	1	ExpMod-CRT	2	ExpMod-ACT4	3	ECC-ADD	4	ExpMod-ACT2	5	ECC-MUL	6	ExpMod-variable	7	ModInv
0	None																	
1	ExpMod-CRT																	
2	ExpMod-ACT4																	
3	ECC-ADD																	
4	ExpMod-ACT2																	
5	ECC-MUL																	
6	ExpMod-variable																	
7	ModInv																	
11 (R/W)	CPY	<p>Copy.</p> <p>Setting the <code>PKA_FUNC.CPY</code> bit performs the copy operation. For more information, see the "PKCP Vector Operations" section.</p>																
10 (R/W)	CMP	<p>Perform Compare Operation.</p> <p>Setting the <code>PKA_FUNC.CMP</code> bit performs the compare operation. For more information, see the "PKCP Vector Operations" section.</p>																

Table 34-32: PKA_FUNC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	MODULO	Perform Modulo Operation. Setting the PKA_FUNC . MODULO bit performs the modulo operation. For more information, see the "PKCP Vector Operations" section.
8 (R/W)	DIV	Perform Divide Operation. Setting the PKA_FUNC . DIV bit performs the divide operation. For more information, see the "PKCP Vector Operations" section.
7 (R/W)	LSHFT	Perform Left Shift Operation. Setting the PKA_FUNC . LSHFT bit performs the Left shift operation. For more information, see the "PKCP Vector Operations" section.
6 (R/W)	RSHFT	Perform Right Shift Operation. Setting the PKA_FUNC . RSHFT bit performs the right shift operation. For more information, see the "PKCP Vector Operations" section.
5 (R/W)	SUB	Perform Subtract Operation. Setting the PKA_FUNC . SUB bit performs the subtract operation. For more information, see the "PKCP Vector Operations" section.
4 (R/W)	ADD	Perform Add Operation. Setting the PKA_FUNC . ADD bit performs the add operation. For more information, see the "PKCP Vector Operations" section.
3 (R/W)	MSONE	Most Significant One. Setting the PKA_FUNC . MSONE bit loads the location of the Most Significant one bit within the result word indicated in the PKA_RESULTMSW register into bits [4:0] of the PKA_DIVMSW register can only be used with basic PKCP operations, except for Divide, Modulo and Compare.
1 (R/W)	ADDSUB	Perform Combined Add/Subtract Operation. Setting the PKA_FUNC . ADDSUB bit performs the combined Add/Subtract operation. For more information, see the "PKCP Vector Operations" section.
0 (R/W)	MULT	Perform Multiply Operation. Setting the PKA_FUNC . MULT bit performs the multiply operation. For more information, see the "PKCP Vector Operations" section.

Start of PKA RAM space

The `PKA_RAM` register provides the the starting location of the RAM space to hold the input, output and other vectors.

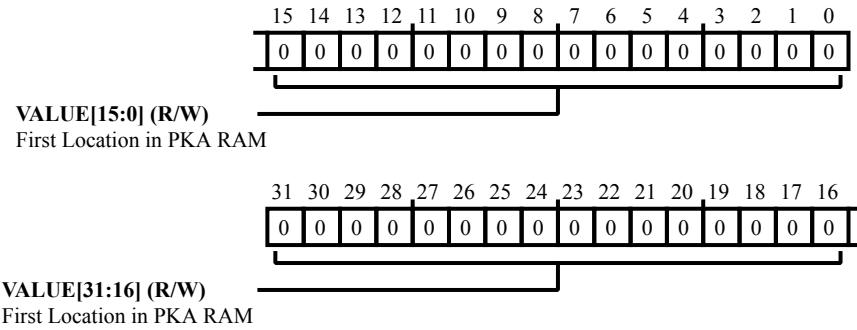


Figure 34-11: PKA_RAM Register Diagram

Table 34-33: PKA_RAM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	First Location in PKA RAM. The <code>PKA_RAM.VALUE</code> bit field provides the the starting location of the RAM space to hold the input, output and other vectors.

PKA Most-Significant-Word of Result Vector

The `PKA_RESULTMSW` register indicates the (word) address in the PKA RAM where the most significant non-zero 32-bit word of the result is stored and should be ignored for modulo operations. For basic PKCP operations, the `PKA_RESULTMSW` register is updated when the `PKA_FUNC.RUN` bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating the final value matching the actual result is done near the end of the operation. Note that the result is only meaningful if no errors are detected and that for ECC operations, the `PKA_DIVMSW` register provides information for the x-coordinate of the result point only.

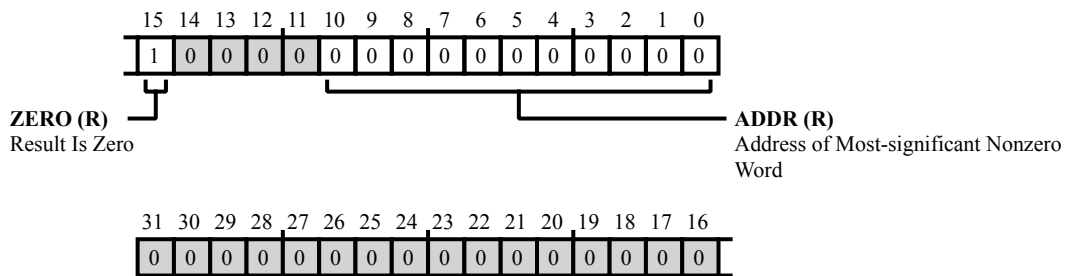


Figure 34-12: `PKA_RESULTMSW` Register Diagram

Table 34-34: `PKA_RESULTMSW` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Result Is Zero. The <code>PKA_RESULTMSW.ZERO</code> bit indicates the result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The <code>PKA_RESULTMSW.ADDR</code> bit is the address of the most significant non-zero 32-bit word of the result vector in PKA RAM.

PKA Bit Shift Value

For basic PKCP operations, modifying the contents of the `PKA_SHIFT` register is made impossible while the operation is being performed. For the ExpMod-variable and ExpMod-CRT operations, the `PKA_SHIFT` register is used to indicate the number of odd powers to use (directly as a value in the range 1-16). For the ModInv and ECC operations, this register is used to hold a completion code.

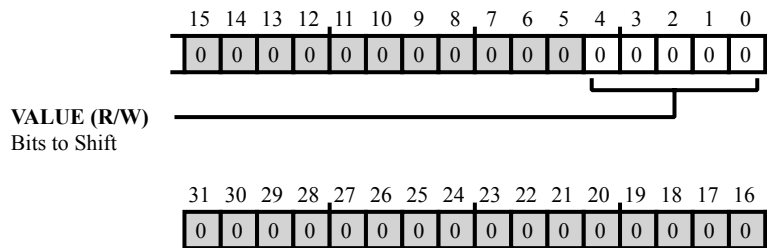


Figure 34-13: `PKA_SHIFT` Register Diagram

Table 34-35: `PKA_SHIFT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Bits to Shift. The <code>PKA_SHIFT.VALUE</code> bit field is the number of bits to shift the input vector (in the range 0-31) during a Rshift or Lshift operation.

35 Public Key Interrupt Controller (PKIC)

The Public Key Accelerator (PKA) and the True Random Number Generator (TRNG) share a common interrupt controller, the Public Key Interrupt Controller (PKIC). The host processor configures the PKIC to generate interrupts when certain operations are complete or interrupts are caused by errors.

PKIC Functional Description

The main purpose and function of the PKIC is to capture the interrupts from different sources, either from the PKA or the TRNG and combine them to one interrupt output. The interrupt controller is managed using the following register groups:

- Control for polarity, edge, and level detection and enabling of individual interrupts
- Acknowledgment (to clear edge detected interrupts)
- Status:
 - A raw source status register after edge detection, if edge selected.
 - A status register after masking with the interrupt enable control bits.

ADSP-2156x PKIC Register List

The Public Key Processor Interrupt Controller (PKIC) provides security-related features. A set of registers governs PKIC operations. For more information on PKIC functionality, see the PKIC register descriptions.

Table 35-1: ADSP-2156x PKIC Register List

Name	Description
<code>PKIC_ACK</code>	Acknowledge Register
<code>PKIC_EN_CLR</code>	Enable Clear Register
<code>PKIC_EN_CTL</code>	Enable Control Register
<code>PKIC_EN_SET</code>	Enable Set Register
<code>PKIC_EN_STAT</code>	Enabled Status Register
<code>PKIC_POL_CTL</code>	Polarity Control Register

Table 35-1: ADSP-2156x PKIC Register List (Continued)

Name	Description
PKIC_RAW_STAT	Raw Status Register
PKIC_TYPE_CTL	Type Control Register

ADSP-2156x PKIC Interrupt List

Table 35-2: ADSP-2156x PKIC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
110	PKIC0_IRQ	PKIC0 Interrupt	Level	

PKIC Programming Model

The following sections provide information on how to program the PKIC.

Enabling/Disabling and Status

The [PKIC_EN_STAT](#) register provides the mask to which interrupt source is enabled. There are two status registers, [PKIC_RAW_STAT](#) and [PKIC_EN_STAT](#). They allow the host processor to read the status of the interrupt source before and after the mask is applied.

Level or Edge

All of the interrupt sources are level or edge events. The [PKIC_TYPE_CTL](#) register configures each interrupt to either level or edge. The [PKIC_POL_CTL](#) register controls the polarity of the signal.

These interrupts are latched at both status registers in case edge detection is selected. The edge detectors are reset by clearing the interrupts using the [PKIC_ACK](#) registers.

PKIC Programming Concepts

The following concepts help with proper programming for the PKIC module.

Interrupt Handling

When an interrupt is triggered, the handler must first examine this module, the PKIC to determine what triggered the interrupt. By reading the [PKIC_EN_STAT](#), the bits that are set are the pending interrupts of the ones that were not masked. After determining the source of the interrupt, the appropriate action must be taken to service the interrupt from the corresponding module, the PKA, or the TRNG. After handling the interrupt in a particular module, the corresponding interrupt must be acknowledged and cleared in the PKIC to allow further interrupts.

While handling an interrupt, any events that would cause another interrupt would happen without triggering another interrupt.

Overlapping Registers

There are two sets of overlapping registers in the PKIC. The `PKIC_EN_STAT` and `PKIC_ACK` registers share one address. If read, the register tells which enabled interrupts are pending. If written to (W1C), the interrupt is acknowledged and cleared. The `PKIC_RAW_STAT` and `PKIC_EN_SET` registers are another pair that share the address. When read, the register tells which interrupts are pending and if written to will enable certain interrupts. This register cannot be used to disable any interrupts.

ADSP-2156x PKIC Register Descriptions

Public Key Processor Interrupt Controller (PKIC) contains the following registers.

Table 35-3: ADSP-2156x PKIC Register List

Name	Description
<code>PKIC_ACK</code>	Acknowledge Register
<code>PKIC_EN_CLR</code>	Enable Clear Register
<code>PKIC_EN_CTL</code>	Enable Control Register
<code>PKIC_EN_SET</code>	Enable Set Register
<code>PKIC_EN_STAT</code>	Enabled Status Register
<code>PKIC_POL_CTL</code>	Polarity Control Register
<code>PKIC_RAW_STAT</code>	Raw Status Register
<code>PKIC_TYPE_CTL</code>	Type Control Register

Acknowledge Register

The `PKIC_ACK` register is used to acknowledge the interrupt and clear the corresponding interrupt bit in the status register.

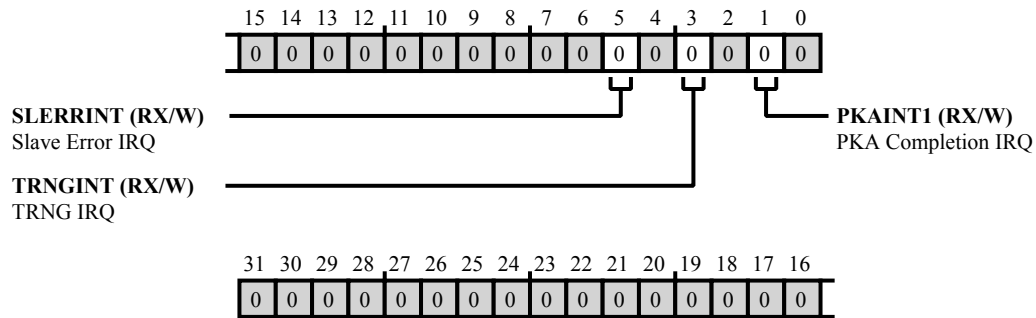


Figure 35-1: PKIC_ACK Register Diagram

Table 35-4: PKIC_ACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_ACK.SLERRINT</code> bit is the acknowledge bit for the Slave Error interrupt. When set =1 the <code>PKIC_ACK.SLERRINT</code> bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).
		0 Do not acknowledge interrupt and clear status bit
		1 Acknowledge interrupt and clear status bit
3 (RX/W)	TRNGINT	TRNG IRQ. The <code>PKIC_ACK.TRNGINT</code> bit is the acknowledge bit for the TRNG interrupt. When set =1 the <code>PKIC_ACK.TRNGINT</code> bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).
		0 Do not acknowledge interrupt and clear status bit
		1 Acknowledge interrupt and clear status bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_ACK.PKAINT1</code> bit is the acknowledge bit for the PKA completion interrupt. When set =1 the <code>PKIC_ACK.PKAINT1</code> bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).

Enable Clear Register

The `PKIC_EN_CLR` register allows the user to disable certain interrupts without enabling others. The disabled interrupts are also reflected in `PKIC_EN_CTL` register.

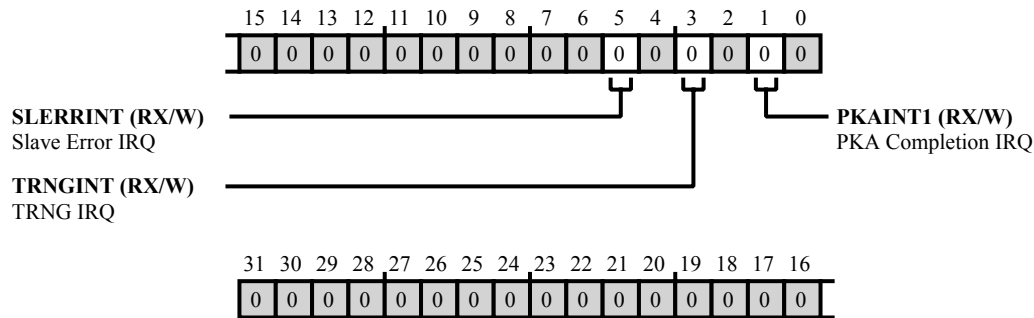


Figure 35-2: `PKIC_EN_CLR` Register Diagram

Table 35-5: `PKIC_EN_CLR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_EN_CLR.SLERRINT</code> bit is the individual disable for the Slave Error interrupt. When set =1 this bit clears/resets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 0 (disables the interrupt). This bit is cleared automatically.
		0 No action
		1 Clear/reset corresponding CTL bit
3 (RX/W)	TRNGINT	TRNG IRQ. The <code>PKIC_EN_CLR.TRNGINT</code> bit is the individual disable for the TRNG interrupt. When set =1 this bit clears/resets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.
		0 No action
		1 Clear/reset corresponding CTL bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_EN_CLR.PKAINT1</code> bit is the individual disable for the PKA Completion interrupt. When set =1 this bit clears/resets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.
		0 No action
		1 Clear/reset corresponding CTL bit

Enable Control Register

The `PKIC_EN_CTL` register provides individual enable bits for the interrupt sources.

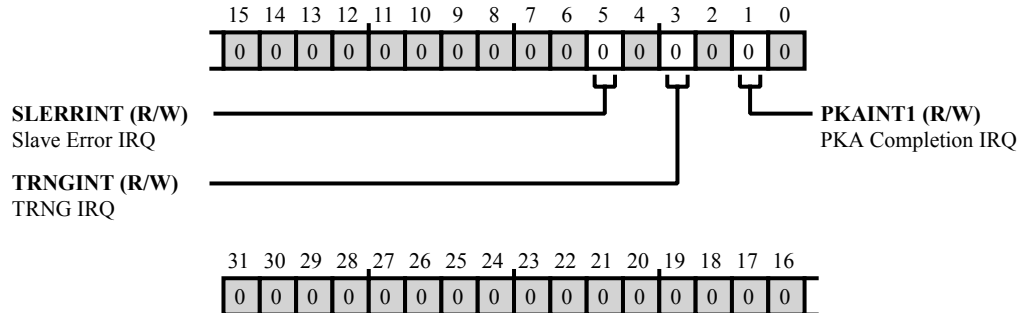


Figure 35-3: `PKIC_EN_CTL` Register Diagram

Table 35-6: `PKIC_EN_CTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_EN_CTL.SLERRINT</code> bit enables control for the Slave Error interrupt.
		0 Disable interrupt
		1 Enable interrupt
3 (R/W)	TRNGINT	TRNG IRQ. The <code>PKIC_EN_CTL.TRNGINT</code> bit enables control for the TRNG interrupt.
		0 Disable interrupt
		1 Enable interrupt
1 (R/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_EN_CTL.PKAINT1</code> bit enables control for the PKA completion interrupt.
		0 Disable interrupt
		1 Enable interrupt

Enable Set Register

The `PKIC_EN_SET` register allows the user to only set certain interrupt sources without disabling any others. The enabled interrupts are reflected in `PKIC_EN_CTL` register.

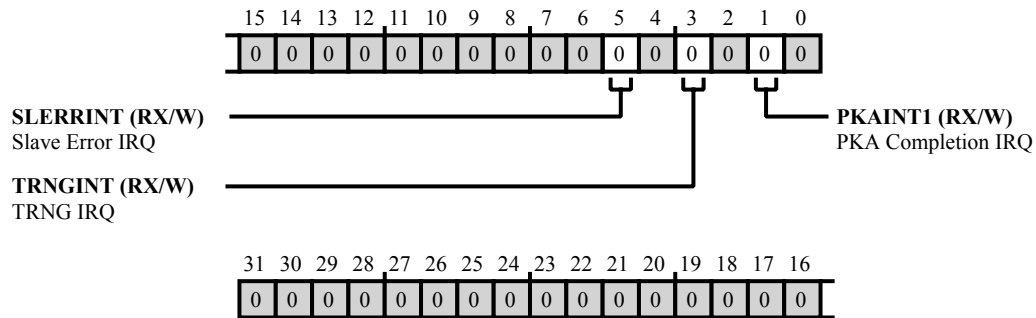


Figure 35-4: `PKIC_EN_SET` Register Diagram

Table 35-7: `PKIC_EN_SET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_EN_SET.SLERRINT</code> bit is the individual enable for the Slave Error interrupt. If =1, sets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 1 (enables interrupt). This bit is cleared automatically. 0=no effect
		0 No effect
		1 Enable interrupt
3 (RX/W)	TRNGINT	TRNG IRQ. The <code>PKIC_EN_SET.TRNGINT</code> bit is the individual enable for the TRNG interrupt. If =1, sets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 1 (enables interrupt). This bit is cleared automatically.
		0 No effect
		1 Enable interrupt
1 (RX/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_EN_SET.PKAINT1</code> bit is the individual enable for the PKA Completion interrupt. If =1, sets the corresponding bit in the <code>PKIC_EN_CTL</code> register to 1 (enables interrupt). This bit is cleared automatically.
		0 No effect
		1 Enable interrupt

Enabled Status Register

The `PKIC_EN_STAT` register is used to tell the status of the interrupts after the gating with the `PKIC_EN_CTL` register.

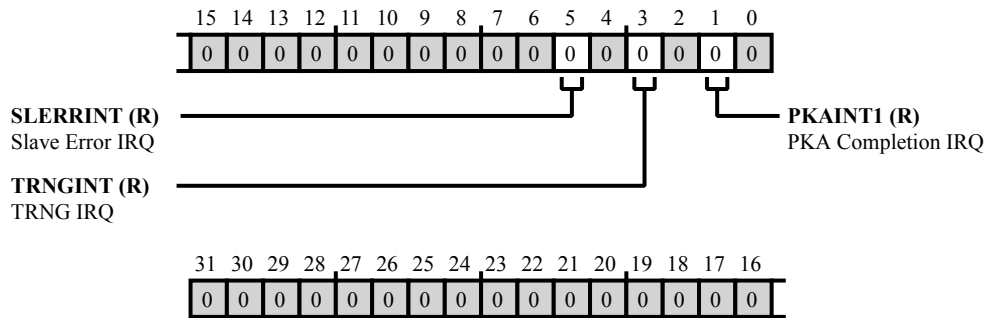


Figure 35-5: `PKIC_EN_STAT` Register Diagram

Table 35-8: `PKIC_EN_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	SLERRINT	Slave Error IRQ. The <code>PKIC_EN_STAT.SLERRINT</code> bit provides the status of the Slave Error interrupt (after masking from the <code>PKIC_EN_CTL</code> register).
		0 Interrupt is inactive
		1 Interrupt is pending
3 (R/NW)	TRNGINT	TRNG IRQ. The <code>PKIC_EN_STAT.TRNGINT</code> bit provides the status of the TRNG interrupt (after masking from the <code>PKIC_EN_CTL</code> register).
		0 Interrupt is inactive
		1 Interrupt is pending
1 (R/NW)	PKAINT1	PKA Completion IRQ. The <code>PKIC_EN_STAT.PKAINT1</code> bit provides the status of the PKA Completion interrupt (after masking from the <code>PKIC_EN_CTL</code> register).
		0 Interrupt is inactive
		1 Interrupt is pending

Polarity Control Register

The `PKIC_POL_CTL` register is used to configure the signal polarity for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to (high level/rising edge or low level/falling edge).

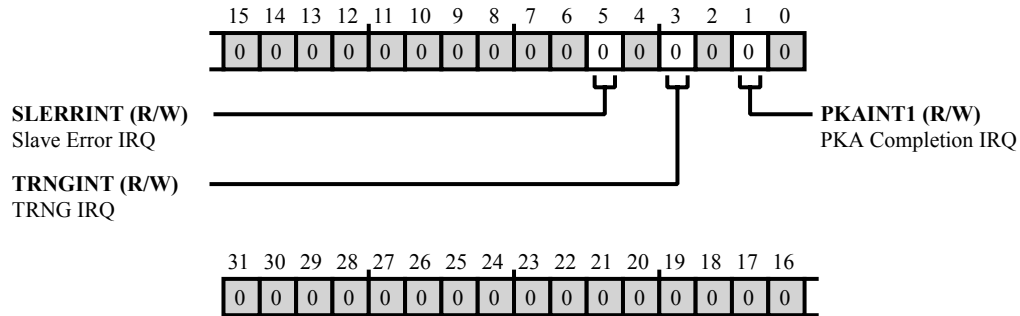


Figure 35-6: PKIC_POL_CTL Register Diagram

Table 35-9: PKIC_POL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_POL_CTL.SLERRINT</code> bit provides polarity control for the Slave Error interrupt.
		0 Low level/falling edge
		1 High level/rising edge
3 (R/W)	TRNGINT	TRNG IRQ. The <code>PKIC_POL_CTL.TRNGINT</code> bit provides polarity control for the TRNG interrupt.
		0 Low level/falling edge
		1 High level/rising edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_POL_CTL.PKAINT1</code> bit provides polarity control for PKA completion interrupt.
		0 Low level/falling edge
		1 High level/rising edge

Raw Status Register

The `PKIC_RAW_STAT` register reflects the status of the individual interrupts before masking with the `PKIC_EN_CTL` register.

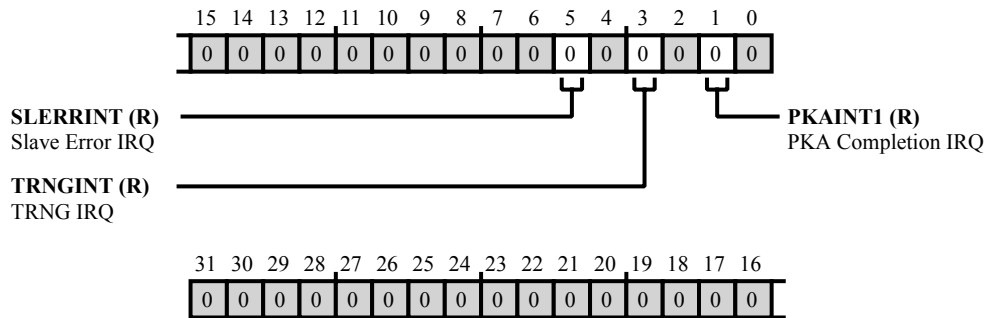


Figure 35-7: `PKIC_RAW_STAT` Register Diagram

Table 35-10: `PKIC_RAW_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	SLERRINT	Slave Error IRQ. The <code>PKIC_RAW_STAT.SLERRINT</code> bit provides the raw status of the Slave Error interrupt.
		0 Inactive interrupt
		1 Pending interrupt
3 (R/NW)	TRNGINT	TRNG IRQ. The <code>PKIC_RAW_STAT.TRNGINT</code> bit provides the raw status of the TRNG interrupt where 1=pending and 0=inactive.
		0 Inactive interrupt
		1 Pending interrupt
1 (R/NW)	PKAINT1	PKA Completion IRQ. The <code>PKIC_RAW_STAT.PKAINT1</code> bit provides raw status of the PKA completion interrupt where 1=pending and 0=inactive.
		0 Inactive interrupt
		1 Pending interrupt

Type Control Register

The `PKIC_TYPE_CTL` register is used to configure the signal type for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to level or edge.

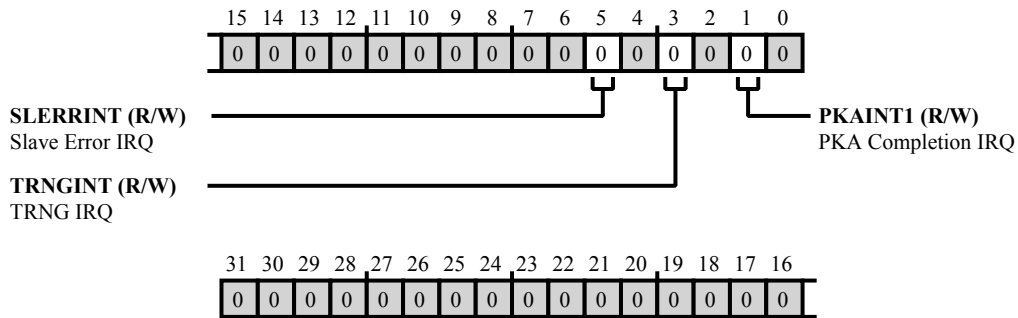


Figure 35-8: `PKIC_TYPE_CTL` Register Diagram

Table 35-11: `PKIC_TYPE_CTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SLERRINT	Slave Error IRQ. The <code>PKIC_TYPE_CTL.SLERRINT</code> bit provides signal type control for the Slave Error interrupt.
		0 Level
		1 Edge
3 (R/W)	TRNGINT	TRNG IRQ. The <code>PKIC_TYPE_CTL.TRNGINT</code> bit provides signal type control for the TRNG interrupt.
		0 Level
		1 Edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The <code>PKIC_TYPE_CTL.PKAINT1</code> bit provides signal type control for the PKA completion interrupt.
		0 Level
		1 Edge

36 True Random Number Generator (TRNG)

The TRNG engine provides a true, non-deterministic, noise source for generating keys, Initialization Vectors (IVs), and other random number requirements. Other non-cryptographic purposes include statistical sampling, retry timers for communications protocols and noise generation.

TRNG Features

The TRNG features include:

- Hardware-based non-deterministic random number generator
- ANSI X9.31 postprocessing (depending on processor)
- Redundant 'fail-safe' design with self-test circuits
- Reliable shot noise oscillator implementation with auto-tuning
- Debug output to allow monitoring of internal operation
- Alarm count overflow and auto-tuning error interrupts
- Buffer to allow generation of large blocks of random data in the background

TRNG Functional Description

The following sections provide details on the function of the TRNG module.

ADSP-2156x TRNG Register List

The True Random Number Generator (TRNG) provides random number generation, intended mainly for security-related applications. A set of registers governs TRNG operations. For more information on TRNG functionality, see the TRNG register descriptions.

Table 36-1: ADSP-2156x TRNG Register List

Name	Description
TRNG_ALMCNT	TRNG Alarm Counter Register

Table 36-1: ADSP-2156x TRNG Register List (Continued)

Name	Description
TRNG_ALMMSK	TRNG Alarm Mask Register
TRNG_ALMSTP	TRNG Alarm Stop Register
TRNG_BLKCNT	TRNG Block Count Register
TRNG_CFG	TRNG Configuration Register
TRNG_CNT	Counter Register
TRNG_CTL	TRNG Control Register
TRNG_FRODETUNE	TRNG FRO De-tune Register
TRNG_FROEN	TRNG FRO Enable Register
TRNG_INPUT[n]	TRNG Input Registers
TRNG_INTACK	TRNG Interrupt Acknowledge Register
TRNG_KEY[n]	Post-Process Key Registers
TRNG_LFSR_H	TRNG LFSR Access Register
TRNG_LFSR_L	TRNG LFSR Access Register
TRNG_LFSR_M	TRNG LFSR Access Register
TRNG_MONOBITCNT	TRNG Monobit Test Result Register
TRNG_OUTPUT[n]	TRNG Output Registers
TRNG_POKER[n]	TRNG Poker Test Result Registers
TRNG_RUNCNT	TRNG Run Count Registers
TRNG_RUN[n]	TRNG Run Test State and Result Registers
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register
TRNG_V[n]	TRNG Post-Process "V" Value Registers

Random Number Generation

The random numbers that the TRNG generates are produced by sampling Free Running Oscillators (FRO). The (TRNG_CTL.STARTUPCYC) bit field along with TRNG_CFG.MINREFCYC (minimum refill cycles) and TRNG_CFG.MAXREFCYC (maximum refill cycles) bit fields determine the number of samples taken to generate the first random value and subsequent random values.

1. After enabling the TRNG (TRNG_CTL.TRNGEN bit =1), a number of FRO output samples defined by the TRNG_CTL.STARTUPCYC bit field are gathered in the main linear-feedback shift register (LFSR) before taking a snapshot of the LFSR and storing that snapshot in the random data buffer (after optional post-processing).

2. After taking a snapshot of the LFSR, a number of FRO output samples defined by the `TRNG_CFG.MINREFCYC` bit field, are gathered in the main LFSR. If the random data buffer is full, sample-taking continues until the number of samples (counting from the snapshot) matches the number of samples defined by the `TRNG_CFG.MAXREFCYC` bit field. At that point, the TRNG switches off the FROs and powers down.
3. If, after the `TRNG_CFG.MINREFCYC` sampling period, the random data buffer is not completely filled, a new snapshot of the LFSR is taken and stored in the random data buffer (after optional postprocessing). Control branches back to point step 2 above and a new `TRNG_CFG.MINREFCYC` sample period starts.

Locking Detection and Prevention

Lock detection in functional mode uses the sampled outputs of the individual FROs. A FRO alarm event is declared when a repeating pattern (of up to four sample lengths) is detected continuously for the number of samples defined in the alarm threshold `TRNG_ALMCNT.ALMTHRESH` bit field. The alarm event is logged by setting the bit that corresponds to the FRO that caused the alarm in the `TRNG_ALMMSK` register. If that bit was already set, the corresponding bit in the `TRNG_ALMSTP` register is set. The FRO is switched off to prevent further alarm events from that FRO. If the `TRNG_ALMMSK` register bit was not yet set, the FRO is restarted automatically in an attempt to break locking.

The shutdown count field in the alarm count `TRNG_ALMCNT.SHDNCNT` register monitors the number of FROs switched off. (It counts the number of 1 bits in the `TRNG_ALMSTP` register.) The shutdown threshold field (`TRNG_ALMCNT.SHDNTHRESH`) can be configured to generate the shutdown overflow interrupt (`TRNG_STAT.SHDNOVR`). When the shutdown count in the `TRNG_ALMCNT.SHDNCNT` bit field exceeds the shutdown threshold in the `TRNG_ALMCNT.SHDNTHRESH` bit field, the shutdown overflow bit (`TRNG_STAT.SHDNOVR`) is set to 1 (which can be used to generate an interrupt).

Software can use two strategies for the TRNG operation:

- **Monitored Operation.** Software checks the `TRNG_ALMMSK` register at regular intervals (on the order of seconds). If a bit is set in that register, then the program must also check the `TRNG_ALMSTP` register to determine if a FRO was shut down due to multiple alarm events. If no FROs are shut down, the program clears the `TRNG_ALMMSK` register to remove the incidental alarm events. If one or more FROs are shut down, the host processor can modify the delay selection of those FROs using the `TRNG_FRODETUNE` register to prevent further locking. For this type of operation, the shutdown threshold is normally set to a low value (two, for example). The shutdown overflow interrupt can then be used to signal abnormal operation conditions or the breakdown of FROs.
- **Unmonitored Operation.** Software sets the shutdown threshold to the acceptable number of FROs to be shut down before taking corrective actions. It then uses the shutdown overflow interrupt to initiate corrective actions (clearing the `TRNG_ALMMSK` and `TRNG_ALMSTP` registers, toggling bits in the `TRNG_FRODETUNE` register). The software must monitor the time interval between these interrupts. If they occur too often (for example, within a minute after each other), this frequency indicates abnormal operating conditions or the breakdown of FROs.

Run Testing

Run Test

The TRNG block counts the number of consecutive zeros and ones (runs) in the data stream shifted into the main LFSR. The run length and bit value is then used to increment a specific bucket counter for these values. After 20,000 bits, the bucket counters must be within specified limits for this test to pass. If not, a run fail interrupt (RUNFAIL) is generated.

Table 36-2: Allowable Limits on Runs of 0's and 1's

Run Count	Bit Value	Min (inclusive)	Max (inclusive)
1	0	2267	2733
	1	2267	2733
2	0	1079	1421
	1	1079	1421
3	0	502	748
	1	502	748
4	0	233	402
	1	233	402
5	0	90	223
	1	90	223
6 and up	0	90	233
	1	90	233

Long Run Test

The long run test fails immediately when a run longer than 33 bits is found and a long run fail (TRNG_STAT.LRUNFAIL) interrupt is generated.

Noise Source Test

A noise source failure is declared when a run of 48 or more identical bits is found and a noise fail interrupt (TRNG_STAT.NOISEFAIL) is generated.

The status and counts are stored in the `TRNG_RUNCNT` and `TRNG_RUN[n]` registers. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the monobit fail acknowledge (TRNG_INTACK.MBITFAIL), the run fail acknowledge (TRNG_INTACK.RUNFAIL), or the poker fail acknowledge (TRNG_INTACK.PKRFail) bits.

Monobit Testing

The TRNG block performs the monobit test on blocks of 20,000 bits (in parallel with the run test and poker test). It monitors the number of zeros and ones in the data stream shifted into the main LFSR. At the start of the block,

the counter is initialized to 10,000. Each 1 value increments the counter, each 0 value decrements the counter. After 20,000 bits, the counter value must be within 9310–10690 (inclusive) for this test to pass. If not, a monobit fail interrupt (`TRNG_STAT.MBITFAIL`) is generated. The AIS-31 standard (test T1, ref 0) specifies this run time testing of the TRNG and the parameters.

NOTE: The actual limits stated here are different than the limits stated in the AIS-31 standard due to the implementation. The circuitry in the TRNG uses an up-down counter while the standard just evaluates the sum of the 20,000 bits.

When the continue poker (`TRNG_TEST.CONTPKR`) bit is set to 1, the test is not stopped after 20,000 bits. The counter keeps incrementing and decrementing (protected against overflow and underflow). But, no actual limit checking happens. The offset from starting value 10,000 indicates the balance of 0 and 1 bits that were checked since the start of the continuous test. The offset is twice the number of missing or extra 1 bits. (An extra 1 bit adds an increment operation and removes one decrement operation. So, having 10,001 1 bits in the block gives a counter value of 10,002.)

Poker Testing

The poker test is run in parallel with the monobit test and run test. Counters in the `TRNG_POKER[n]` registers are used to count the occurrences of one specific 4-bit value in the data stream fed into the main LFSR. All of the counters are decremented by one every 64 data bits and reset to their start value every 20,000 bits. All counters start at a value of -1 and are decremented 312 times during the 20,000-bit test run.

Each 8-bit counter holds a two's complement value and does not overflow past the range of -128 to $+127$. At the end of the 20,000-bits block, the values of the counters that contain a single 1 bit appended at the least significant bit side are individually squared and then added. The poker test fails with a poker fail (`PKRFAIL`) interrupt when:

- the resulting sum (accumulated in the `TRNG_MONOBITCNT` register) is outside the range 1288 – 71750 (inclusive), or
- one of the counters tries to increment or decrement outside its limit range

NOTE: The poker test fails when the 4-bit values of the data stream are distributed too evenly (with eight counters having incremented 312 times and the others incremented 313 times). This failure is intentional. The minimum mean deviation from the expected value of 312.5 is 4.5. Failure at counter overflow is not an official part of the poker test as specified in the AIS-31 standard (ref 0). It can be shown that the maximum deviation for one counter's value from the mean value of 312.5 (without the poker test failing) is 129.5. Since this deviation is more than 40% of the mean value, it indicates that something is wrong. Here, the counter overflow failure is combined with the official poker test failure.

When the continue poker (`TRNG_TEST.CONTPKR`) bit is set to 1, the test does not stop after 20,000 bits. The counters keep incrementing and decrementing (the latter every 64 bits). A `PKRFAIL` interrupt is generated when one of the counters tries to increment or decrement outside its limit range.

Data for Tests

For the monobit test, run test and poker test circuits self-test, the test data written to the `TRNG_INPUT[n]0` register is used for the monobit test and run test bit-by-bit. It executes from bit 31 down to bit 0. For the poker test, the written test data is used in eight blocks of 4 bits, starting from bits 31:28 down to bits 3:0.

When the `TRNG_TEST.CONTPKR` bit =0 during run test and poker test circuits self-tests, the state of the poker and runs test status registers is frozen after all calculations are made and after inputting 20,000 test bits. This state allows time to read the test results. These calculations take around 20 clock cycles to complete. The status registers are reset to their starting states when the first word for the next test block of test bits is written to the `TRNG_INPUT[n]0` register. Then, the contents of this word are processed.

X9.31 Postprocessing

Postprocessing is available on some parts using the TRNG. If available, the postprocessing block is situated after the main LFSRs and before the output buffer that stores the random numbers for consumption. The online test logic for monobit, run, and poker testing is before the postprocessor block. The bits used for testing are from the main LFSR.

The postprocessor is based on the ANSI X9.31 specification using 3-Key 3DES cipher algorithm. It does not provide any more entropy than is what is already achieved from sampling the Free Running Oscillators (FROs). The use of the postprocessor only helps with applications requiring the use of an X9.31 compliant RNG.

The following section is an example of postprocessing.

The variables include:

- DT is a 64-bit date/time vector. This value is the input coming from random bits from the main LFSR.
- I is an intermediate value, a 64-bit value
- R is the final result, a 64-bit value
- V is a 64-bit seed value that is to be kept secret
- K is the 3-key for 3DES, each being 64-bits

The postprocessing uses the steps:

1. The intermediate value I is calculated: $I = 3DES_{EDE}(K, DT)$ with 3DES.
2. The result R is calculated: $R = 3DES_{EDE}(K, I \text{ XOR } V)$.
3. Finally, V is updated: $V = 3DES_{EDE}(K, R \text{ XOR } I)$.

TRNG Block Diagram

The *System Block Diagram of the TRNG* diagram shows the system block for the TRNG. The system includes:

- Free Running Oscillators (FROs) that are the source of sampled bits
- A post-processing unit (processor dependent) for standards compliance

- Test circuitry to detect non-randomness due to failures in the system

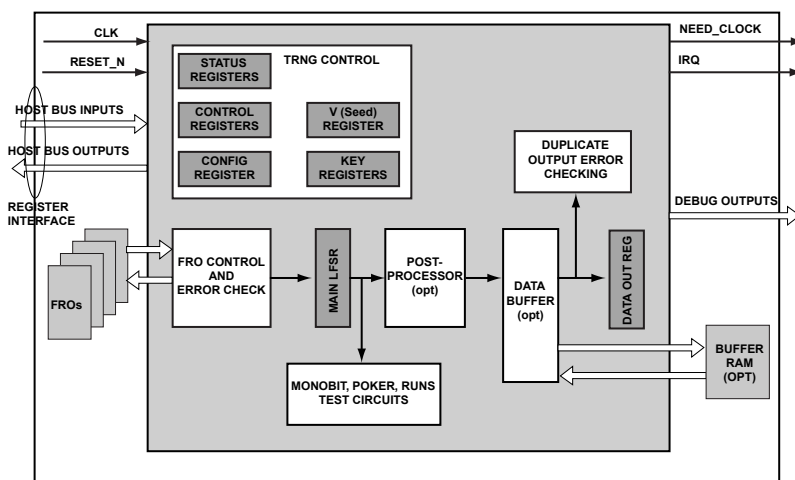


Figure 36-1: System Block Diagram of the TRNG

TRNG Architectural Concepts

The random numbers are accessible to the host processor in four 32-bit registers allowing a single burst read of a 128-bit random number. Acknowledging the data ready (interrupt) state causes the TRNG to move a new value (when available in the data buffer) to the TRNG output register. The TRNG always tries to keep the data buffer completely full. Pulling out data starts the regeneration of a new number by:

- Enabling the FROs
- Capturing their outputs in the LFSR, and
- Cryptographically postprocessing the values from the LFSR

The process produces new random values to replenish the buffer.

The major functional blocks of the TRNG module are:

- The actual TRNG core with control and test circuits, optional post-processor, and optional data buffer control logic
- Free Running Oscillators (FROs) instantiated outside the TRNG core
- A 128-byte buffer RAM instantiated outside the TRNG core

In the TRNG core, the true entropy source uses FROs as the basic building block. The accumulation of timing jitter, caused (for the largest part) by shot noise, creates uncertainty intervals for the output transitions of each FRO. Sampling within an uncertainty interval generates a single bit of entropy, which is ‘accumulated’ in a LFSR. As the uncertainty interval is very narrow compared to the cycle time of a FRO, the mean amount of entropy generated per sample is small (less than 1/100 bit per sample). To increase the entropy generation rate, multiple FROs are used in parallel.

The FROs are asynchronous to one another and asynchronous to the sampling clock to make their behavior truly non-deterministic.

The output signals of the FROs are sampled at regular intervals (in general, at the TRNG core module clock frequency). The samples feed into an error detection circuit that checks for repeating patterns coming out of a FRO. If a repeating pattern persists for a configurable number of samples, the FRO is suspect of having synchronized to (a harmonic of) the sampling interval. This activity drastically reduces the amount of entropy generated by that FRO. The error detection circuit signals this activity as a FRO error event.

Error events can occur during normal operation. The FRO control circuit attempts to restart a FRO that on a first error event. A second error event causes an automatic shutdown of the FRO. Because there are multiple FROs, shutting down a FRO reduces the amount of entropy generated, but it does not immediately jeopardize the TRNG operation. A limit can be configured below which the number of operational FROs does not drop. If this limit is crossed, an interrupt can be generated on the host processor. Software on the host processor can then attempt to prevent frequent locking of a FRO by *de-tuning* it to a slightly different frequency.

An XOR tree combines the sampled outputs and feeds them into an 81-bit LFSR to accumulate entropy and whiten the random bits stream.

NOTE: Here, *whitening* means balancing the number of one and zero bits in the stream.

TRNG Operating Modes

The TRNG has the following operating modes.

Normal Reading Mode

In normal reading mode, random data can only be read out of the output registers of the TRNG module when the ready bit (`TRNG_STAT.RDY`) = 1. Acknowledging the data by writing a 1 to the ready acknowledge bit in the `TRNG_INTACK.RDY` register clears the ready bit from the status register and clears the output registers. The registers remain at zero until the next 128-bit data block is available.

Secure Reading Mode

An attacker can try to read the output registers (without acknowledging the data) to obtain a copy of data to be read later by an application. To block this attack, secure reading mode is used. In this mode, enable reading from the output registers (by writing 0x00 to the open read gate bits (`TRNG_INTACK.OPENRDGATE`)) before it is possible to access the output registers. Enabling the reading starts a timeout (controlled by the `TRNG_CFG.RDTIMEOUT` bit field). When this timeout expires, reading is disabled and the offered data is acknowledged so that it is not offered again. The host processor must set this timeout such that there is enough time to read the output registers and perform a normal data acknowledge (which aborts the timeout).

Test Mode

In addition to the test circuitry that operates during normal operation, the TRNG has a test mode that allows further diagnosis when errors occur. In test mode, programs have access to the main LFSR through the

`TRNG_LFSR_L`, `TRNG_LFSR_M`, and `TRNG_LFSR_H` registers. Programs can also control the finite state machine sample counter using the `TRNG_CNT` register.

In test mode, the TRNG can be configured to test individual or a chosen set of FROs. It can also be configured to feed test patterns to the delay chain.

TRNG Data Transfer Modes

The host processor reads four 32-bit registers to access the random numbers. Once the registers are read and the data ready interrupt has been acknowledged, the TRNG moves more data from the internal buffer to the output registers (`TRNG_OUTPUT[n]`).

TRNG Event Control

There are eight events that the TRNG generates. The events are common error events from the run-time testing of the TRNG. While the TRNG is operating and generating random bits, test circuitry is also running statistical tests. The tests determine if the sources of the random bits have started to fail and are not truly producing random bits. There is also a single event to signal when data is ready in the output registers.

These events are captured in the `TRNG_STAT` register and can generate a single interrupt in the Public Key Interrupt Controller (PKIC). The individual events can be masked so the interrupt is not triggered. This configuration uses the `TRNG_CTL` register. The event and associated interrupt can be acknowledged in the `TRNG_INTACK` register.

The *TRNG Interrupt Signals* table lists all events or interrupts that the TRNG can generate.

TRNG Interrupt Signals

The TRNG provides a total of eight interrupts multiplexed into one output.

Table 36-3: TRNG Interrupt Signals

Interrupt	Name	Description
0	RDY	Ready. Random number is ready to be read from registers
1	SHDNOVR	Shutdown Overflow. The number of FRO's automatically shut down due to failures or errors have gone above the threshold specified in <code>TRNG_ALMCNT.SHDNTHRESH</code> .
2	STUCKOUT	Stuck Out. Logic circuitry around the output registers has detected the same output has been provided twice.
3	NOISEFAIL	Noise Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 48 identical bits, which is considered a noise source failure.
4	RUNFAIL	Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value for at least one of the <code>TRNG_RUN[n]</code> counters after checking 20,000 bits.

Table 36-3: TRNG Interrupt Signals (Continued)

Interrupt	Name	Description
5	LRUNFAIL	Long Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 34 identical bits.
6	PKRFAIL	Poker Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 <code>TRNG_POKER[n]</code> counters or an out-of-bounds sum of squares values after checking 20,000 bits.
7	MBITFAIL	Monobit Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds number of 1's after checking 20,000 bits.

ADSP-2156x TRNG Register Descriptions

True Random Number Generator (TRNG) contains the following registers.

Table 36-4: ADSP-2156x TRNG Register List

Name	Description
<code>TRNG_ALMCNT</code>	TRNG Alarm Counter Register
<code>TRNG_ALMMSK</code>	TRNG Alarm Mask Register
<code>TRNG_ALMSTP</code>	TRNG Alarm Stop Register
<code>TRNG_BLKCNT</code>	TRNG Block Count Register
<code>TRNG_CFG</code>	TRNG Configuration Register
<code>TRNG_CNT</code>	Counter Register
<code>TRNG_CTL</code>	TRNG Control Register
<code>TRNG_FRODETUNE</code>	TRNG FRO De-tune Register
<code>TRNG_FROEN</code>	TRNG FRO Enable Register
<code>TRNG_INPUT[n]</code>	TRNG Input Registers
<code>TRNG_INTACK</code>	TRNG Interrupt Acknowledge Register
<code>TRNG_KEY[n]</code>	Post-Process Key Registers
<code>TRNG_LFSR_H</code>	TRNG LFSR Access Register
<code>TRNG_LFSR_L</code>	TRNG LFSR Access Register
<code>TRNG_LFSR_M</code>	TRNG LFSR Access Register
<code>TRNG_MONOBITCNT</code>	TRNG Monobit Test Result Register
<code>TRNG_OUTPUT[n]</code>	TRNG Output Registers
<code>TRNG_POKER[n]</code>	TRNG Poker Test Result Registers
<code>TRNG_RUNCNT</code>	TRNG Run Count Registers
<code>TRNG_RUN[n]</code>	TRNG Run Test State and Result Registers

Table 36-4: ADSP-2156x TRNG Register List (Continued)

Name	Description
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register
TRNG_V[n]	TRNG Post-Process "V" Value Registers

TRNG Alarm Counter Register

The `TRNG_ALMCNT` register, together with the `TRNG_ALMMSK` and `TRNG_ALMSTP` registers, can be used by the host processor to determine if the FRO/sample cycle locking is a problem. Note that incidental alarm events are expected to occur during normal operation. This register also controls the way the monobit test and poker test circuits operate (using the standard 20,000 bit blocks or running continuously).

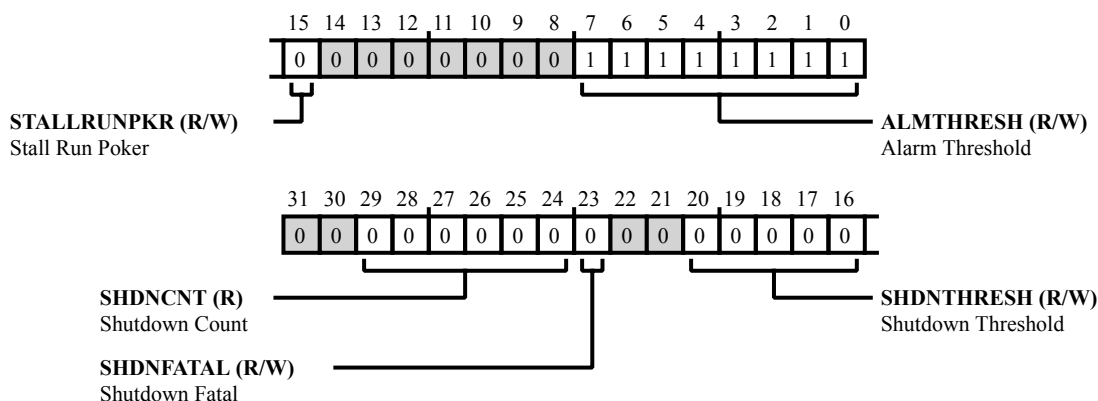


Figure 36-2: TRNG_ALMCNT Register Diagram

Table 36-5: TRNG_ALMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	SHDNCNT	Shutdown Count. This read-only field indicates the number of 1 bits in the <code>TRNG_ALMSTP</code> register, the number of FRO's that's been turned off.
23 (R/W)	SHDNFATAL	Shutdown Fatal. When the <code>TRNG_ALMCNT.SHDNFATAL</code> bit field is set, the shutdown overflow (SHDNOVR) interrupt is considered a fatal error requiring taking the complete TRNG engine off-line.
20:16 (R/W)	SHDNTHRESH	Shutdown Threshold. The <code>TRNG_ALMCNT.SHDNTHRESH</code> bit field provides the threshold setting for generating the shutdown overflow (SHDNOVR) interrupt, which is activated when the shutdown count (<code>TRNG_ALMCNT.SHDNCNT</code>) value in this register exceeds the threshold value set here.
15 (R/W)	STALLRUNPKR	Stall Run Poker. When the <code>TRNG_ALMCNT.STALLRUNPKR</code> bit is set, stalls the Monobit Test, Run Test and Poker Test circuits when either the <code>TRNG_STAT.MBITFAIL</code> , <code>TRNG_STAT.RUNFAIL</code> or <code>TRNG_STAT.PKRFAIL</code> bits =1. This allows inspection of the state of the result counters (which would otherwise be reset immediately for the next 20,000 bits block to test).

Table 36-5: TRNG_ALMCNT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	ALMTHRESH	<p>Alarm Threshold.</p> <p>The TRNG_ALMCNT.ALMTHRESH bit field sets the alarm detection threshold for the repeating pattern detectors on each FRO. A FRO alarm event is declared when a repeating pattern (of up to four samples length) is detected continuously for the number of samples defined by this fields value. Reset value 255 (decimal) should keep the number of alarm events to a manageable level.</p>

TRNG Alarm Mask Register

A set bit (=1) in the `TRNG_ALMMSK` register signifies an alarm event and is used by the host processor to determine which of the individual FROs generated the alarm. If a bit in this register is set, the corresponding bit in the `TRNG_ALMSTP` register is set and the FRO is turned off by clearing the corresponding bit in the `TRNG_FROEN` register. If a bit is not set, the FRO restarts automatically to try to break sample cycle locking that could have caused the alarm event.

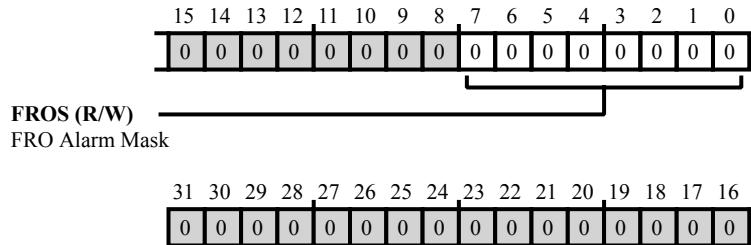


Figure 36-3: TRNG_ALMMSK Register Diagram

Table 36-6: TRNG_ALMMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Mask. The <code>TRNG_ALMMSK.FROS</code> bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced an alarm event.

TRNG Alarm Stop Register

The `TRNG_ALMSTP` register is used by the host processor to determine which of the individual FROs generated more than one alarm event in quick succession. If a FRO generates an alarm event while a previous event is still logged in the `TRNG_ALMMSK` register, the corresponding bit in this register is set (=1) and the FRO is turned off by clearing (=0) the corresponding bit in the `TRNG_FROEN` register. The `TRNG_ALMCNT.SHDNCNT` bit field keeps track of the number of bits that are set in this register.

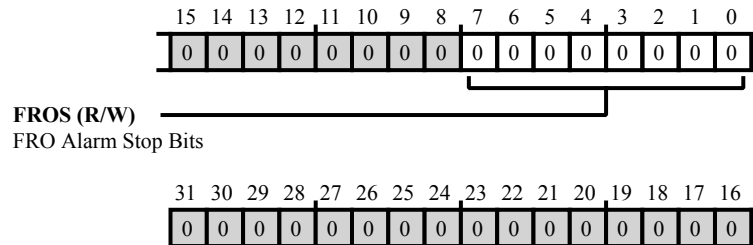


Figure 36-4: TRNG_ALMSTP Register Diagram

Table 36-7: TRNG_ALMSTP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Stop Bits. The <code>TRNG_ALMSTP.FROS</code> bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced more than one alarm event in quick succession and has been turned off. A 1 in this field forces the corresponding bit in the <code>TRNG_FROEN</code> register to 0.

TRNG Block Count Register

The `TRNG_BLKCNT` register is the counter for the 128-bit blocks generated by the post-processor. These bits are forced to zero when the post-processor is disabled and are cleared to zero when an internal re-seed operation has finished. This register can be used by driver software to determine when to re-seed the post-processor.

The whole 32 bits of this register represent the amount of data (in bytes) generated since a re-seed. The `TRNG_BLKCNT` register is only present when a post-processor is available.

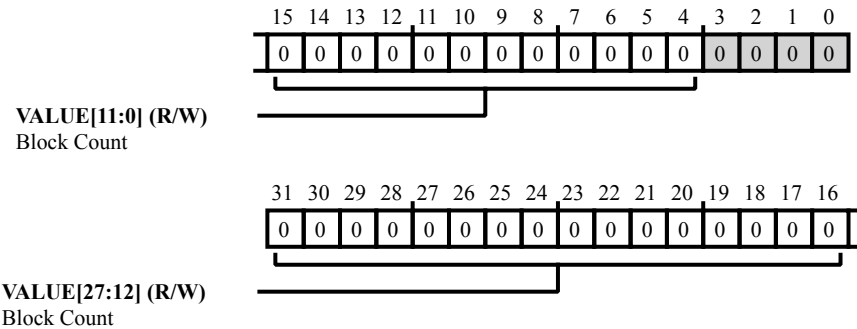


Figure 36-5: TRNG_BLKCNT Register Diagram

Table 36-8: TRNG_BLKCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:4 (R/W)	VALUE	Block Count. The <code>TRNG_BLKCNT.VALUE</code> bit field is the counter for the 128-bit blocks generated by the post-processor. These bits are forced to zero when the post-processor is disabled and are cleared to zero when an internal re-seed operation has finished.

TRNG Configuration Register

The `TRNG_CFG` register holds the lower and upper limits of the samples taken from the FROs in order to refill the random data buffer. This register also holds the time out value used for secure reading mode.

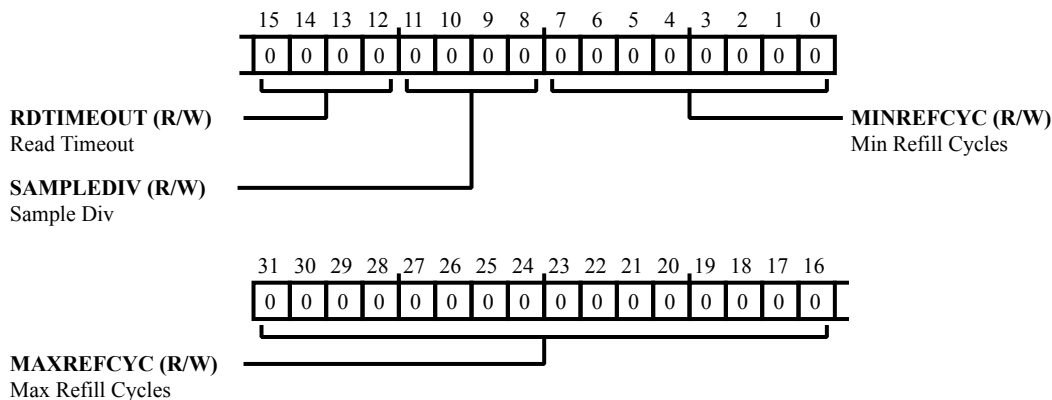


Figure 36-6: TRNG_CFG Register Diagram

Table 36-9: TRNG_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	MAXREFCYC	Max Refill Cycles. The <code>TRNG_CFG.MAXREFCYC</code> bit field determines the maximum number of samples (between 2^8 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number. If the written value of this field is zero, the number of samples is 2^{24} , otherwise the number of samples equals the written value times 2^8 . This field can only be modified while the <code>TRNG_CTL.TRNGEN</code> bit =0.
15:12 (R/W)	RDTIMEOUT	Read Timeout. The <code>TRNG_CFG.RDTIMEOUT</code> bit field controls the Secure Reading Mode. When this field is 0, Secure Reading Mode is disabled. Values in the range 115 enable Secure Reading and set a read gate closure timeout of approximately $(read_timeout + 1) \times 16$ clock input cycles. This field can only be modified while the <code>TRNG_CTL.TRNGEN</code> bit =0.
11:8 (R/W)	SAMPLEDIV	Sample Div. The <code>TRNG_CFG.SAMPLEDIV</code> bit field directly controls the number of input cycles between samples taken from the FROs. The default value 0 indicates that samples are taken every cycle, maximum value 15 (decimal) takes one sample every 16 cycles. This field must be set to a value such that the slowest FRO (even under worst-case conditions) has a cycle time less than twice the sample period. The default configuration of the FROs allows this field to remain 0. This field can only be modified while <code>TRNG_CTL.TRNGEN=0</code> .

Table 36-9: TRNG_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MINREFCYC	<p>Min Refill Cycles.</p> <p>The <code>TRNG_CFG.MINREFCYC</code> bit field determines the minimum number of samples (between 2^6 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number.</p> <p>If the value of this field is zero, the number of samples is fixed to the value determined by the maximum refill cycles (<code>TRNG_CFG.MAXREFCYC</code>) field, otherwise the minimum number of samples equals the written value times 64 (which can be up to 2^{14}). The number of samples defined here cannot be higher than the number defined by the <code>TRNG_CFG.MAXREFCYC</code> field (i.e. that field takes precedence).</p> <p>This field can only be modified while the <code>TRNG_CTL.TRNGEN</code> bit =0.</p>

Counter Register

The `TRNG_CNT` register is used to access the main control Finite State Machine's (FSM) sample counter while the `TRNG_CTL.TSTMODE` bit =1.

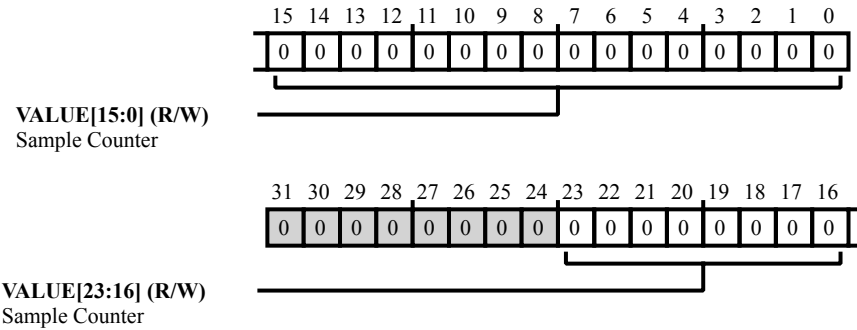


Figure 36-7: TRNG_CNT Register Diagram

Table 36-10: TRNG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Sample Counter. The <code>TRNG_CNT.VALUE</code> bit field is the sample counter used by control finite state machine. This counter can only be accessed when the <code>TRNG_CTL.TSTMODE</code> bit =1.

TRNG Control Register

The `TRNG_CTL` register must be written to start accumulating entropy before random numbers can be generated. In most cases, the `TRNG_CFG` register must also be written prior to writing the `TRNG_CTL` register. To enable the TRNG, set the `TRNG_CTL.TRNGEN` bit. This register also controls post-processing (if available). Note that when the `TRNG_CTL.TRNGEN` bit = 1, the start up cycles field (`TRNG_CTL.STARTUPCYC`) and the post-processing enable bit (if available) are locked. Any writes to these fields are ignored.

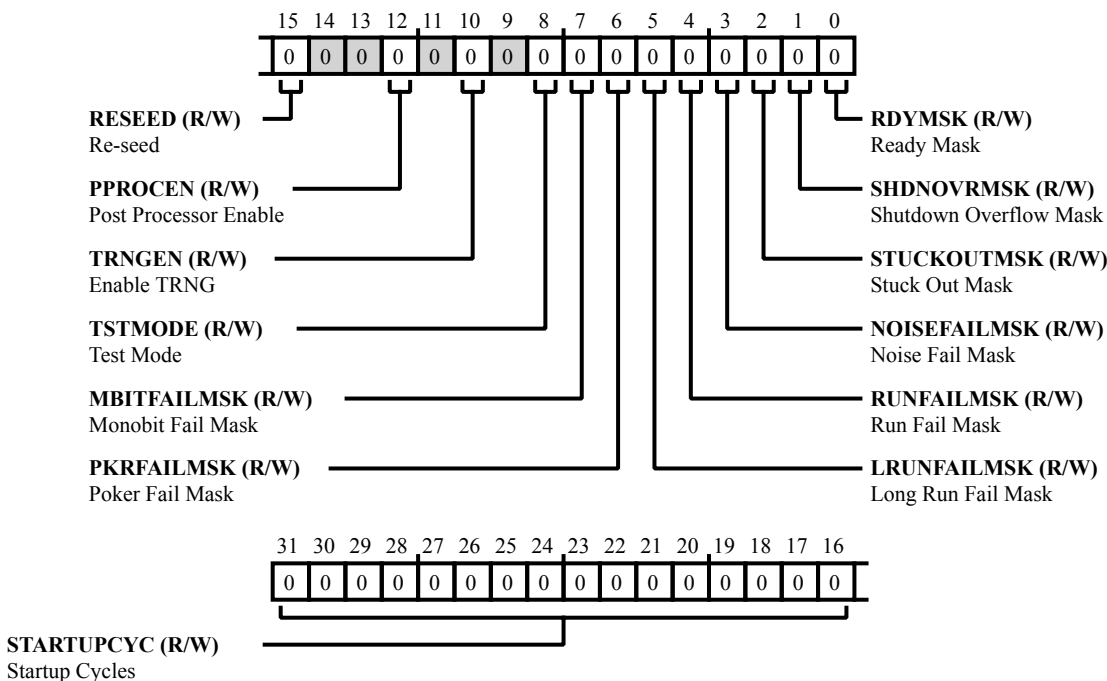


Figure 36-8: TRNG_CTL Register Diagram

Table 36-11: TRNG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	STARTUPCYC	Startup Cycles. The <code>TRNG_CTL.STARTUPCYC</code> bit field determines the number of samples (between 2^8 and 2^{24}) taken to gather entropy from the FROs during startup. If the written value of this field is zero, the number of samples is 2^{24} , otherwise the number of samples equals the written value times 2^8 . This field can only be written when <code>TRNG_CTL.TRNGEN=0</code> before the write.

Table 36-11: TRNG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	RESEED	<p>Re-seed.</p> <p>The TRNG_CTL . RESEED bit is set-only, writing a 1 starts a re-seed cycle and writing a 0 has no effect. A re-seed cycle entails loading the TRNG_KEY[n] and TRNG_V[n] registers with random values generated internally these values are not visible outside the TRNG core.</p> <p>This bit falls back to 0 automatically after the re-seed operation is complete at that time the TRNG_BLKCNT register is reset to zero and the random data buffer is zero-ized so that any new data read from the TRNG will use the new seed values.</p> <p>This bit is only present when post-processing is available and can only be set to 1 when TRNG_CTL . TRNGEN=1 before the write. Note that re-seeding can be done with the post-processor disabled (normally used to seed the post-processor before enabling it).</p> <p>When writing a 1 to this bit, all other bits in this register remain unchanged.</p>
12 (R/W)	PPROCEN	<p>Post Processor Enable.</p> <p>Setting the TRNG_CTL . PPROCEN bit enables the FIPS post-processor. If this bit is reset to 0, the post-processor is forced back into the idle state immediately. This bit is only present when post-processing is available and can only be changed when the TRNG_CTL . TRNGEN bit (enable TRNG) was 0 before the write.</p> <p>To change the TRNG_CTL . PPROCEN bit during operation, first put the TRNG into reset (TRNG_CTL . TRNGEN =0). If the post-processor is enabled, it can be disabled by a subsequent write of 0 to this bit (writing this bit when the TRNG_CTL . TRNGEN bit is still 1 has no effect). The post-processor then stops immediately. To enable it, this bit must be written with 1.</p> <p>Changing the enabled/disabled state does not affect the contents of the the TRNG_KEY[n] and TRNG_V[n] registers. After changing the state, the TRNG must be started again by setting the TRNG_CTL . TRNGEN bit to 1. Note that it is required to re-gather entropy, so the same number of start-up cycles must be used as when starting the TRNG out of a system reset state.</p>
10 (R/W)	TRNGEN	<p>Enable TRNG.</p> <p>Setting the TRNG_CTL . TRNGEN bit to 1 starts the TRNG, gathering entropy from the FROs for the number of samples determined by the value in the TRNG_CTL . STARTUPCYC (Startup Cycles) field. Resetting this bit to 0 forces all TRNG logic back into the idle state immediately. Resetting this bit to 0 also performs the Un-instantiate operation, clearing all internal post-processor registers.</p>
8 (R/W)	TSTMODE	<p>Test Mode.</p> <p>When the TRNG_CTL . TSTMODE bit is set, access is enabled to the TRNG_CNT and TRNG_LFSR_L, TRNG_LFSR_M and TRNG_LFSR_H registers (the latter are cleared before enabling access) and sets the TRNG_STAT . NEEDCLK bit for testing purposes. This bit must be set to 1 before various test modes in the TRNG_TEST register can be enabled.</p>

Table 36-11: TRNG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	MBITFAILMSK	Monobit Fail Mask. When the TRNG_CTL.MBITFAILMSK bit is set, this mask allows the TRNG_STAT.MBITFAIL bit to activate the (active HIGH) interrupt output.
6 (R/W)	PKRFAILMSK	Poker Fail Mask. When the TRNG_CTL.PKRFAILMSK bit is set, this mask allows the TRNG_STAT.PKRFAIL bit to activate the (active HIGH) interrupt output.
5 (R/W)	LRUNFAILMSK	Long Run Fail Mask. When the TRNG_CTL.LRUNFAILMSK bit is set, this mask allows the TRNG_STAT.LRUNFAIL bit to activate the (active HIGH) interrupt output.
4 (R/W)	RUNFAILMSK	Run Fail Mask. When the TRNG_CTL.RUNFAILMSK bit is set, this mask allows the TRNG_STAT.RUNFAIL bit to activate the (active HIGH) interrupt output.
3 (R/W)	NOISEFAILMSK	Noise Fail Mask. When the TRNG_CTL.NOISEFAILMSK bit is set, this mask allows the TRNG_STAT.NOISEFAIL bit to activate the (active HIGH) interrupt output.
2 (R/W)	STUCKOUTMSK	Stuck Out Mask. When the TRNG_CTL.STUCKOUTMSK bit is set, this mask allows the TRNG_STAT.STUCKOUT bit to activate the (active HIGH) interrupt output.
1 (R/W)	SHDNOVRMSK	Shutdown Overflow Mask. When the TRNG_CTL.SHDNOVRMSK bit is set, this mask allows the TRNG_STAT.SHDNOVR bit to activate the (active HIGH) interrupt output.
0 (R/W)	RDYMSK	Ready Mask. When the TRNG_CTL.RDYMSK bit is set, this mask allows the TRNG_STAT.RDY bit to activate the (active HIGH) interrupt output.

TRNG FRO De-tune Register

The `TRNG_FRODETUNE` register is used by the host processor to change the frequencies of individual FROs. This can reduce the number of alarm events generated by a specific FRO.

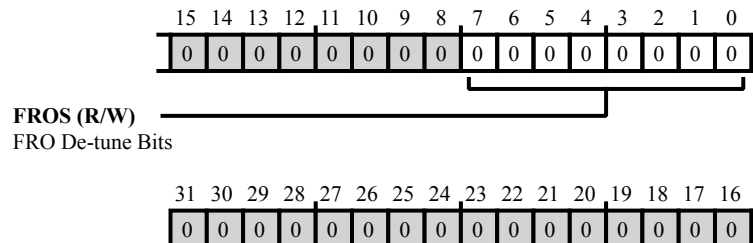


Figure 36-9: TRNG_FRODETUNE Register Diagram

Table 36-12: TRNG_FRODETUNE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	<p>FRO De-tune Bits.</p> <p>The <code>TRNG_FRODETUNE.FROS</code> bits De-tune the FROs. A 1 in bit [n] lets FRO n run approximately 5% faster. The value of one of these bits may only be changed while the corresponding FRO is turned off (by temporarily writing a 0 in the corresponding bit of the <code>TRNG_FROEN</code> register).</p>

TRNG FRO Enable Register

The `TRNG_FROEN` register can be used by the host processor to enable and disable FROs individually. Only enabled FROs contribute to entropy generation, but require power to do so. Disabled FROs cannot generate alarm events.

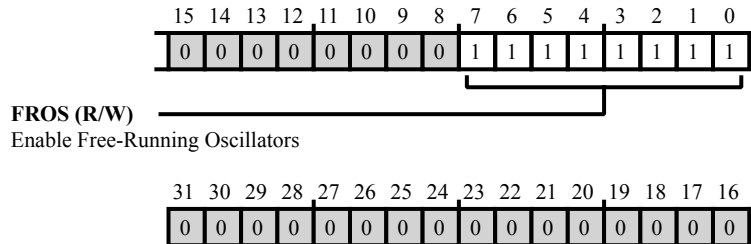


Figure 36-10: TRNG_FROEN Register Diagram

Table 36-13: TRNG_FROEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	<p>Enable Free-Running Oscillators.</p> <p>The <code>TRNG_FROEN.FROS</code> bits are the enables for the individual FROs. A 1 in bit [n] enables FRO n. The default state is all ones to enable all FROs after power-up. Note that the FROs are not actually started up before the <code>TRNG_CTL.TRNGEN</code> bit is set to 1. These bits are automatically forced to 0 (and cannot be written to 1) when the corresponding bit in the <code>TRNG_ALMSTP</code> register has value 1.</p>

TRNG Input Registers

The `TRNG_INPUT[n]` registers are used as input for post-processor testing (if post processing is available) and as input for Monobit Test, Run Test and Poker Test functionality tests (`TRNG_INPUT0` only). They share their addresses with the corresponding `TRNG_OUTPUT[n]` registers. The least significant word is contained in the `TRNG_INPUT0` register.

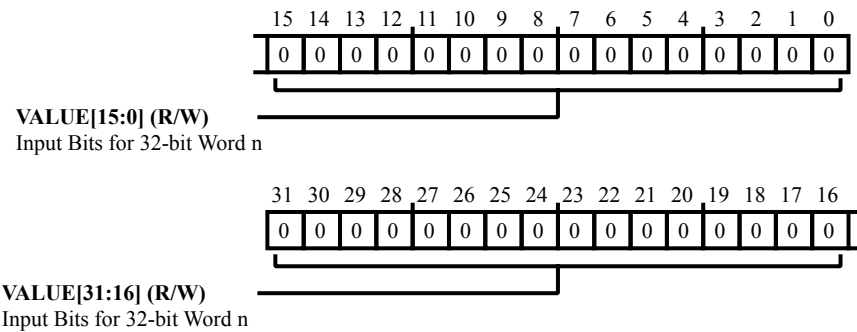


Figure 36-11: `TRNG_INPUT[n]` Register Diagram

Table 36-14: `TRNG_INPUT[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Input Bits for 32-bit Word n. The <code>TRNG_INPUT[n].VALUE</code> bit field is used to hold 32-bits of the 64-bit word of test data for 3-DES post-processor (if available). Or, the <code>TRNG_INPUT[n].VALUE</code> bit field is used to hold 32-bit data word for Run Test and Poker Test circuits self test. Can only be written to when the <code>TRNG_STAT.TSTRDY</code> bit =1.

TRNG Interrupt Acknowledge Register

The `TRNG_INTACK` register is written to acknowledge interrupts indicated in bits [7:0] of the `TRNG_STAT` register. Writing a 1 to any of the bits [7:2] has side effects in resetting various parts of the TRNG core logic which can also be used even if no interrupts are actually active.

When acknowledging the interrupts, these bits are write '1' to clear the associated bit in `TRNG_STAT` register. The bit in this register will also automatically be reset to zero.

When secure reading mode is enabled, write bits [7:0] of this register with zeros to enable TRNG data reads from the `TRNG_OUTPUT[n]` registers. Writing bits [7:0] also starts the (configurable) timeout counter that automatically acknowledges the TRNG data (and disables reads) if the `TRNG_INTACK.RDY` bit is not written with a 1 within that timeout period.

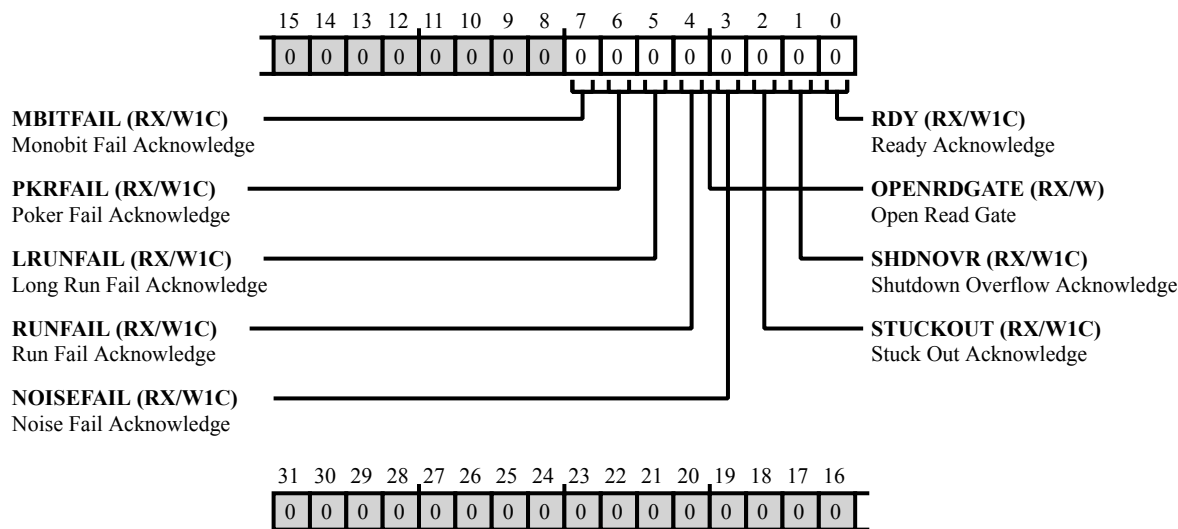


Figure 36-12: TRNG_INTACK Register Diagram

Table 36-15: TRNG_INTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (RX/W1C)	MBITFAIL	Monobit Fail Acknowledge. Set the <code>TRNG_INTACK.MBITFAIL</code> bit to acknowledge the Monobit Fail Interrupt. This also resets all counter and state bits in the <code>TRNG_RUN[n]</code> , <code>TRNG_MONOBITCNT</code> and the <code>TRNG_POKER[n]</code> registers (except for the <code>TRNG_RUNCNT.LENMAX</code> field).
6 (RX/W1C)	PKRFAIL	Poker Fail Acknowledge. Set the <code>TRNG_INTACK.PKRFAIL</code> bit to acknowledge the Poker Fail Interrupt. This also resets all counter and state bits in the <code>TRNG_RUN[n]</code> , <code>TRNG_MONOBITCNT</code> and the <code>TRNG_POKER[n]</code> registers (except for the <code>TRNG_RUNCNT.LENMAX</code> field).

Table 36-15: TRNG_INTACK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W1C)	LRUNFAIL	Long Run Fail Acknowledge. Set the TRNG_INTACK.LRUNFAIL bit to acknowledge the Long Run Fail Interrupt. Also clears the TRNG_RUNCNT.LENMAX field.
4 (RX/W1C)	RUNFAIL	Run Fail Acknowledge. Set the TRNG_INTACK.RUNFAIL bit to acknowledge the Run Fail Interrupt. Also resets all counter and state bits in the TRNG_RUN[n], TRNG_MONOBITCNT and the TRNG_POKER[n] registers (except for the TRNG_RUNCNT.LENMAX field).
3 (RX/W1C)	NOISEFAIL	Noise Fail Acknowledge. Set the TRNG_INTACK.NOISEFAIL bit to acknowledge the Noise Fail Interrupt. Setting this bit also clears the TRNG_RUNCNT.LENMAX (Run Length Max) field, the random data buffer, the TRNG_OUTPUT[n] registers and the TRNG_STAT.RDY bit.
2 (RX/W1C)	STUCKOUT	Stuck Out Acknowledge. Set the TRNG_INTACK.STUCKOUT bit to acknowledge the Stuck Out Interrupt. Setting this bit also clears the random data buffer, the TRNG_OUTPUT[n] registers and the TRNG_STAT.RDY bit.
1 (RX/W1C)	SHDNOVR	Shutdown Overflow Acknowledge. Set the TRNG_INTACK.SHDNOVR bit to acknowledge the Shutdown Overflow Interrupt.
0 (RX/W1C)	RDY	Ready Acknowledge. The TRNG_INTACK.RDY bit allows a new number (if it is ready in the random data buffer), to directly move into the result register. Once done, the TRNG_STAT.RDY bit is reset, after at most size clock cycles.
7:0 (RX/W)	OPENRDGATE	Open Read Gate. In Secure Reading Mode, the TRNG_INTACK.OPENRDGATE bit writes an all zeros value to bits [7:0] to enable reading of TRNG data from the TRNG_OUTPUT[n] registers. This starts the timeout counter that automatically acknowledges the TRNG data (and disables reading) if the TRNG_INTACK.RDY bit is not written with a 1 within that timeout period.

Post-Process Key Registers

The `TRNG_KEY[n]` registers are used to load the key used for post-processing (if available). These registers are write-only. Reads return the values of the other registers mapped at the same addresses.

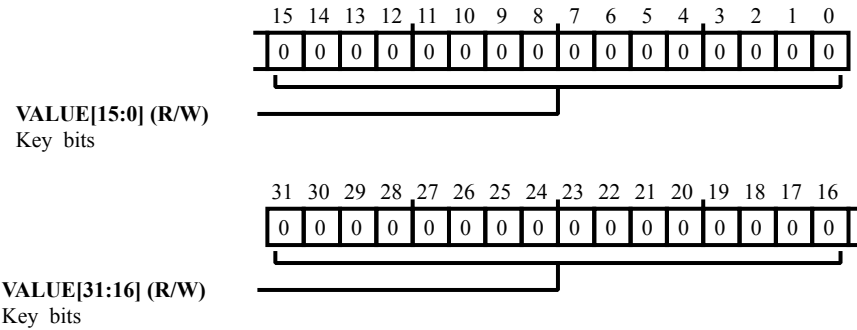


Figure 36-13: `TRNG_KEY[n]` Register Diagram

Table 36-16: `TRNG_KEY[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Key bits. Bits for cipher key used in the post-processor.

TRNG LFSR Access Register

The `TRNG_LFSR_H` register is used to access bits [80:64] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE = 1`).

For security reasons, the LFSR contents are zeroed before enabling access.

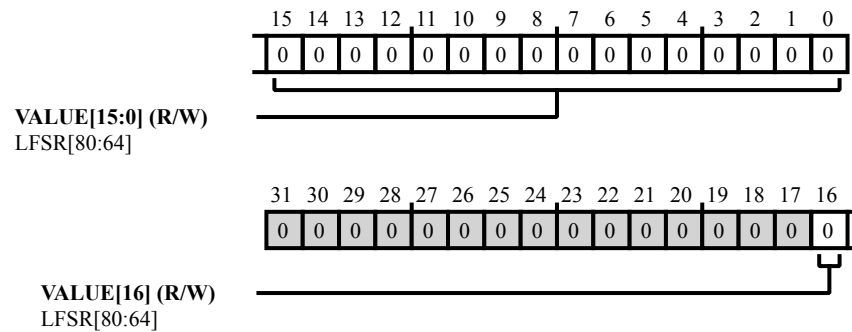


Figure 36-14: TRNG_LFSR_H Register Diagram

Table 36-17: TRNG_LFSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/W)	VALUE	LFSR[80:64]. The <code>TRNG_LFSR_H.VALUE</code> bit field contains bits [80:64] of the main entropy accumulation LFSR. This field can only be accessed when the <code>TRNG_CTL.TSTMODE</code> bit = 1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The `TRNG_LFSR_L` register is used to access bits [31:0] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE = 1`).

For security reasons, the LFSR contents are zeroed before enabling access.

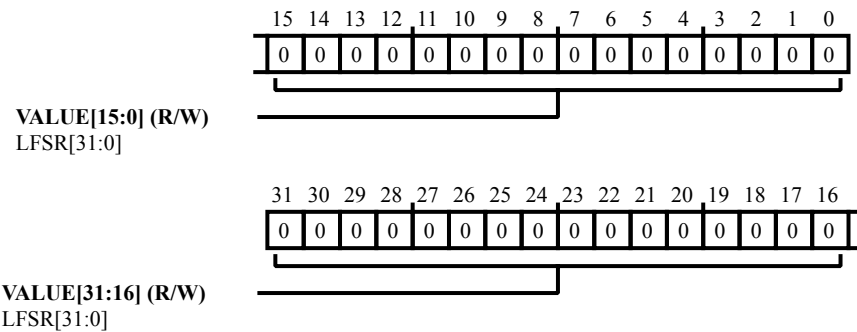


Figure 36-15: TRNG_LFSR_L Register Diagram

Table 36-18: TRNG_LFSR_L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[31:0]. The <code>TRNG_LFSR_L.VALUE</code> bit field contains bits [31:0] of the main entropy accumulation LFSR. This field can only be accessed when the <code>TRNG_CTL.TSTMODE</code> bit =1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The `TRNG_LFSR_M` register is used to access bits [63:32] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE = 1`).

For security reasons, the LFSR contents are zeroed before enabling access.

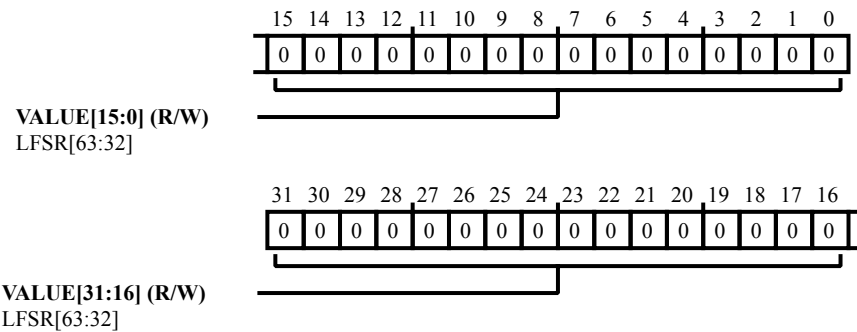


Figure 36-16: TRNG_LFSR_M Register Diagram

Table 36-19: TRNG_LFSR_M Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[63:32]. The <code>TRNG_LFSR_M.VALUE</code> bit field contains bits [63:32] of the main entropy accumulation LFSR. This field can only be accessed when the <code>TRNG_CTL.TSTMODE</code> bit =1. Contents are cleared (=0) before access is enabled.

TRNG Monobit Test Result Register

The `TRNG_MONOBITCNT` register accesses the counter used to perform a Monobit Test as specified by the AIS-31 standard (test T1, ref 4). This test is performed on blocks of 20,000 bits (in parallel to the run test and Poker Test).

Note: Immediately after performing the actual Monobit Test at the end of the 20,000 bits block, the counter is used to accumulate the Poker Test results. As a result, the actual Monobit Test count result value can only be read in the `TRNG_MONOBITCNT` register if the test fails and the stall run Poker (`TRNG_ALMCNT.STALLRUNPKR`) bit = 1.

The monobit test result register is read-only; writing it accesses the registers mapped at the same address. The counter in this register is reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), run fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

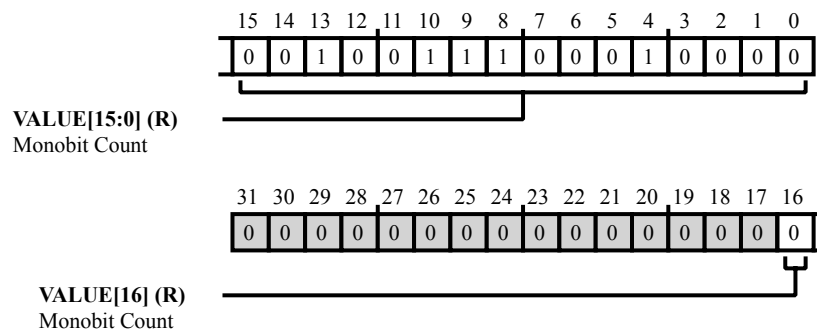


Figure 36-17: `TRNG_MONOBITCNT` Register Diagram

Table 36-20: `TRNG_MONOBITCNT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/NW)	VALUE	Monobit Count. The <code>TRNG_MONOBITCNT.VALUE</code> bit field is the up/down counter which monitors 1 and 0 bits. After 20,000 bits, this counter should have a value in the range 9310 through 10690 (inclusive) to pass the Monobit Test. This counter is protected against overflow and underflow.

TRNG Output Registers

The `TRNG_OUTPUT[n]` registers provide read access to the 128-bit random number output. A subset of these registers are also used as output for post-processor testing (if available). They share their addresses with the `TRNG_INPUT0` through `TRNG_INPUT3` registers. The least significant word is contained in the `TRNG_OUTPUT0` register.

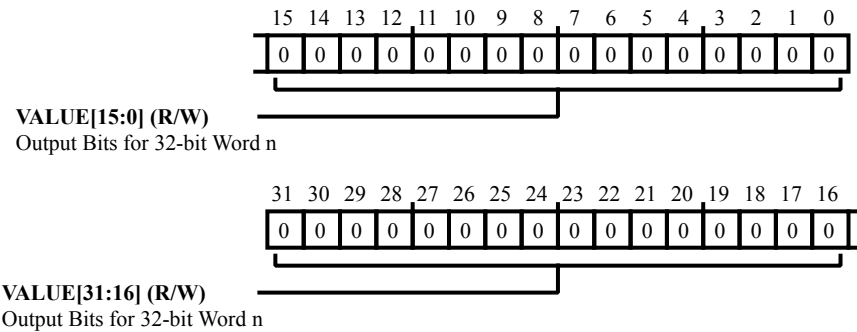


Figure 36-18: `TRNG_OUTPUT[n]` Register Diagram

Table 36-21: `TRNG_OUTPUT[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Output Bits for 32-bit Word n. The <code>TRNG_OUTPUT[n].VALUE</code> bit field is used to holds 32 bits of the 128-bit word of random data. Only valid when the <code>TRNG_STAT.RDY</code> bit =1. Alternatively, this register holds the 32-bits of the 42-bit word of result data for 3-DES post-processing testing. Only valid when the <code>TRNG_STAT.TSTRDY</code> bit =1.

TRNG Poker Test Result Registers

The `TRNG_POKER[n]` registers are used to access the 16 counters used perform a poker test on blocks of 20,000 bits (in parallel to the monobit and run tests).

Poker test result registers are read-only; writing them accesses the registers mapped at these same addresses. All counters in these registers are reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), run fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

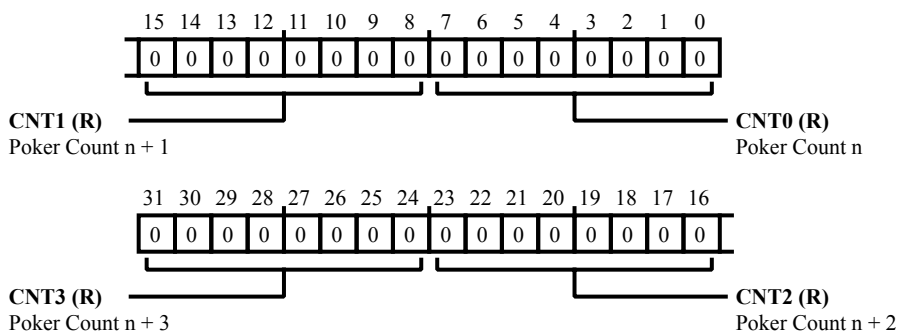


Figure 36-19: `TRNG_POKER[n]` Register Diagram

Table 36-22: `TRNG_POKER[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	CNT3	Poker Count n + 3. The <code>TRNG_POKER[n].CNT3</code> bit field provides the counter for 4-bit value 0x3, 0x7, 0xB, 0xF in <code>TRNG_POKER0</code> , <code>TRNG_POKER1</code> , <code>TRNG_POKER2</code> , <code>TRNG_POKER3</code> , respectively.
23:16 (R/NW)	CNT2	Poker Count n + 2. The <code>TRNG_POKER[n].CNT2</code> bit field provides the counter for 4-bit value 0x2, 0x6, 0xA, 0xE in <code>TRNG_POKER0</code> , <code>TRNG_POKER1</code> , <code>TRNG_POKER2</code> , <code>TRNG_POKER3</code> , respectively.
15:8 (R/NW)	CNT1	Poker Count n + 1. The <code>TRNG_POKER[n].CNT1</code> bit field provides the counter for 4-bit value 0x1, 0x5, 0x9, 0xD in <code>TRNG_POKER0</code> , <code>TRNG_POKER1</code> , <code>TRNG_POKER2</code> , <code>TRNG_POKER3</code> , respectively.
7:0 (R/NW)	CNT0	Poker Count n. The <code>TRNG_POKER[n].CNT0</code> bit field provides the counter for 4-bit value 0x0, 0x4, 0x8, 0xC in the <code>TRNG_POKER0</code> , <code>TRNG_POKER1</code> , <code>TRNG_POKER2</code> , <code>TRNG_POKER3</code> , respectively.

TRNG Run Count Registers

The `TRNG_RUNCNT` registers are used to access the 10 counters that perform a run test and long run test as specified by the AIS-31 standard (tests T3 and T4, ref 4). They are also used to perform the noise source failure test proposed in section E.5 of that same standard.

The `TRNG_RUNCNT` registers are read-only; writing them accesses the other registers which are mapped at the same addresses. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the Monobit Fail acknowledge (`TRNG_INTACK.MBITFAIL`), Run Fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the Poker Fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

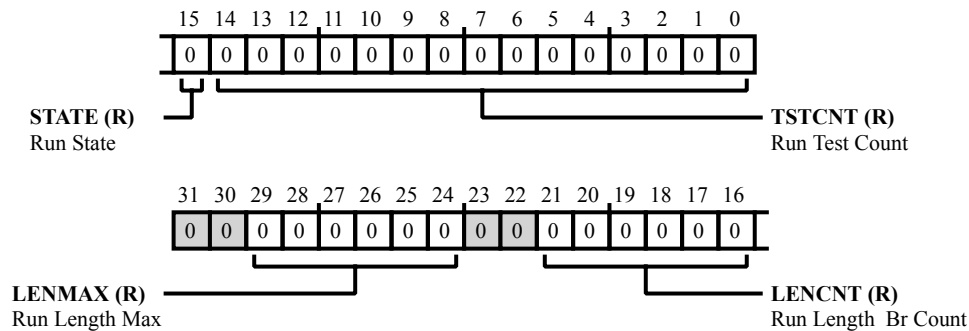


Figure 36-20: TRNG_RUNCNT Register Diagram

Table 36-23: TRNG_RUNCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	LENMAX	Run Length Max. The <code>TRNG_RUNCNT.LENMAX</code> bit field configures the maximum run length count value encountered since start of test. This value is reset back to zero when writing a 1 to either the Noise Fail acknowledge (<code>TRNG_INTACK.NOISEFAIL</code>) or the Long Run Fail acknowledge (<code>TRNG_INTACK.LRUNFAIL</code>) bits.
21:16 (R/NW)	LENCNT	Run Length Br Count. The <code>TRNG_RUNCNT.LENCNT</code> bit field configures the counter for the current run of consecutive 0/1 bits; cannot increment past its maximum value of 63.
15 (R/NW)	STATE	Run State. The <code>TRNG_RUNCNT.STATE</code> bit field provides the state of bits in the current run.
14:0 (R/NW)	TSTCNT	Run Test Count. The <code>TRNG_RUNCNT.TSTCNT</code> bit field configures the block length counter for the run and poker tests - counts up for 20,000 tested bits and then controls testing of the <code>run_X_count...</code> and <code>poker_count_X</code> counters to contain expected values, after which they - and this counter - are reset for the next block.

TRNG Run Test State and Result Registers

The `TRNG_RUN[n]` registers holds the counts for the associated run bucket for 1's and 0's.

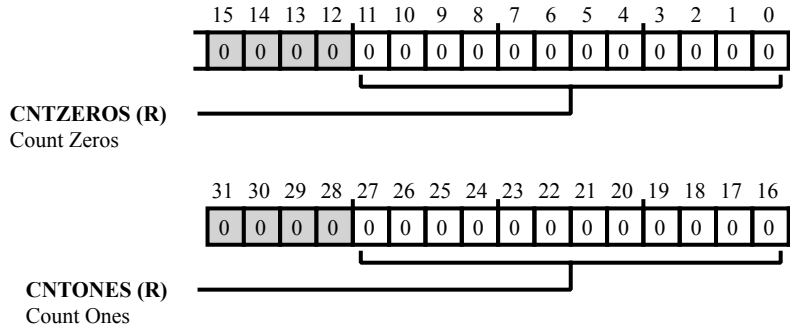


Figure 36-21: TRNG_RUN[n] Register Diagram

Table 36-24: TRNG_RUN[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:16 (R/NW)	CNTONES	<p>Count Ones.</p> <p>In TRNG_RUN1, this counter is for single bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095.</p> <p>In TRNG_RUN2, this counter is for two bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047.</p> <p>In TRNG_RUN3, this counter is for three bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023.</p> <p>In TRNG_RUN4, this counter for four bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511.</p> <p>In TRNG_RUN5, this counter is for five bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.</p> <p>In TRNG_RUN6, this counter for six and higher bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.</p>

Table 36-24: TRNG_RUN[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/NW)	CNTZEROS	<p>Count Zeros.</p> <p>In TRNG_RUN1, this counter is for single bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095.</p> <p>In TRNG_RUN2, this counter is for two bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047.</p> <p>In TRNG_RUN3, this counter is for three bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023.</p> <p>In TRNG_RUN4, this counter is for four bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511.</p> <p>In TRNG_RUN5, this counter is for five bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.</p> <p>In TRNG_RUN6, this counter is for six and higher bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.</p>

TRNG Status Register

The `TRNG_STAT` register provides status results. This register shares the same address as the Interrupt Acknowledge (`TRNG_INTACK`) register.

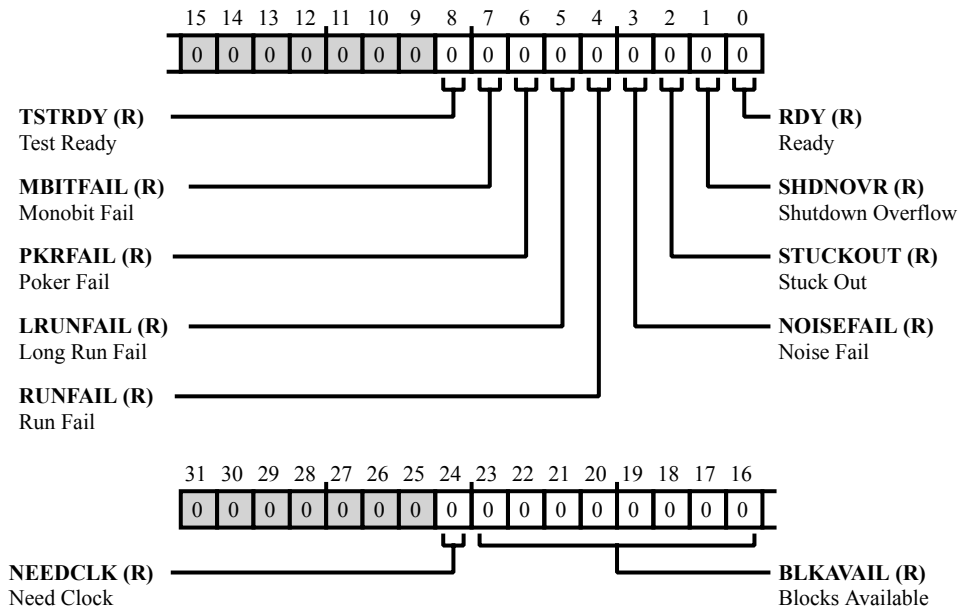


Figure 36-22: TRNG_STAT Register Diagram

Table 36-25: TRNG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/NW)	NEEDCLK	Need Clock. When the <code>TRNG_STAT.NEEDCLK</code> bit is set, it indicates that the TRNG is busy generating entropy or is in one of its test modes the module clock may not be turned off.
23:16 (R/NW)	BLKAVAIL	Blocks Available. This field indicates the number of 128 bits blocks of random data that are available in the random data buffer. If this value is non-zero, the output registers will be re-filled from the random data buffer immediately after acknowledging the <code>TRNG_STAT.RDY</code> by writing a '1' to <code>TRNG_INTACK.RDY</code> .
8 (R/NW)	TSTRDY	Test Ready. When the <code>TRNG_STAT.TSTRDY</code> bit is set, it indicates that data for known-answer tests on the Monobit Test, Run Test, Poker Test and post-processor functions can be written to the <code>TRNG_INPUT[n]</code> registers. When testing the post-processor, result data can be read from those same registers when this bit has become 1 again (after dropping to 0).

Table 36-25: TRNG_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	MBITFAIL	Monobit Fail. When the TRNG_STAT.MBITFAIL bit is set, the Monobit Test logic monitoring data, shifted into the main LFSR, detected an out-of-bounds number of 1s after checking 20,000 bits (test T1 as specified in the AIS-31 standard).
6 (R/NW)	PKRFAIL	Poker Fail. When the TRNG_STAT.PKRFAIL bit is set, the Poker Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 Poker Counters or an out of bounds sum of squares value after checking 20,000 bits (test T2 as specified in the AIS-31 standard).
5 (R/NW)	LRUNFAIL	Long Run Fail. When the TRNG_STAT.LRUNFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 34 identical bits (test T4 as specified in the AIS-31 standard).
4 (R/NW)	RUNFAIL	Run Fail. When the TRNG_STAT.RUNFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value for at least one of the TRNG_RUN[n].CNTZEROS or TRNG_RUN[n].CNTONES counters after checking 20,000 bits (test T3 as specified in the AIS-31 standard).
3 (R/NW)	NOISEFAIL	Noise Fail. When the TRNG_STAT.NOISEFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 48 identical bits, which is considered a noise source failure as proposed in section E.5 of the AIS-31 standard.
2 (R/NW)	STUCKOUT	Stuck Out. When the TRNG_STAT.STUCKOUT bit is set, the logic around the output data registers detected that the TRNG generates the same value twice in a row.
1 (R/NW)	SHDNOVR	Shutdown Overflow. When the TRNG_STAT.SHDNOVR bit is set, the number of FROs shut down after a second error event (the number of 1 bits in the TRNG_ALMSTP register) has exceeded the threshold set by the TRNG_ALMCNT.SHDNTHRESH bit field.
0 (R/NW)	RDY	Ready. When the TRNG_STAT.RDY bit is set, data is available in the TRNG_OUTPUT0 to TRNG_OUTPUT3 registers. If a new number is already available in the random data buffer, that number is directly moved into the result register. In this case the ready status bit is asserted again, after at most six module clock cycles.

TRNG Test Register

The `TRNG_TEST` register can be used by the host processor to perform a number of tests on the TRNG logic including:

- Register controlled characterization by connecting the `tst_fro_clk_out` output to a selected FRO clock output
- FRO logic connectivity and error event detection checking by feeding known patterns through the FRO delay line and error event detection circuits
- Direct XOR-ed FRO outputs capture by disabling the main LFSR feedback logic
- Extend the Monobit Test and Poker Test by not resetting the Monobit count and Poker Test X counters after each 20,000 bits block
- Perform known answer tests on the Run Test, Poker Test and post-processor functions.

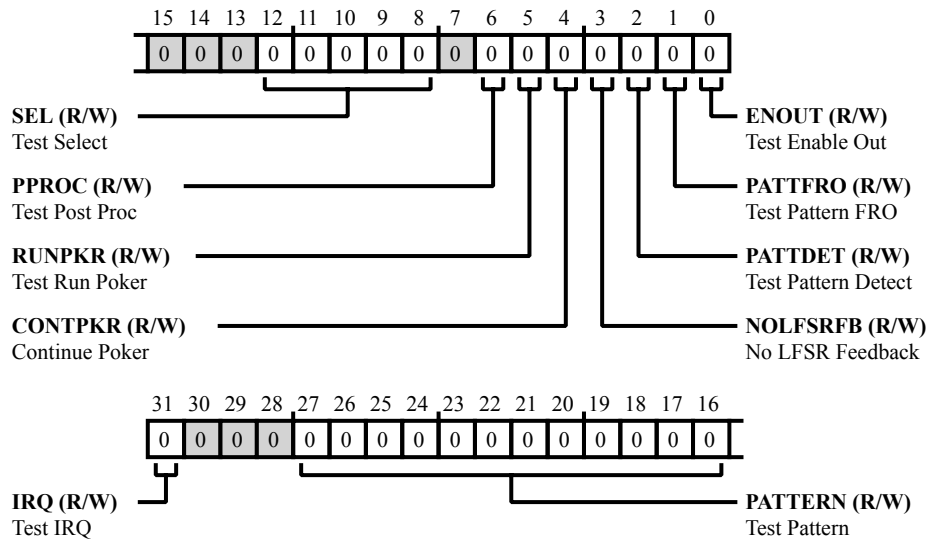


Figure 36-23: TRNG_TEST Register Diagram

Table 36-26: TRNG_TEST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	IRQ	Test IRQ. When the <code>TRNG_TEST . IRQ</code> bit is set force irq output HIGH for interrupt signal connectivity testing.
		0 Do not force IRQ high
		1 Force IRQ output high

Table 36-26: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration				
27:16 (R/W)	PATTERN	<p>Test Pattern.</p> <p>The TRNG_TEST.PATTERN bit field sets up a repeating sequence of bits to be fed into the selected FRO delay chain TRNG_TEST.PATTFRO =1 and/or the selected FRO error detection circuit TRNG_TEST.PATDET =1. This field is rotated right over one bit, once every sample period, when either of these control bits is 1. Therefore, bit [16] is the actual pattern bit fed into the test target.</p>				
12:8 (R/W)	SEL	<p>Test Select.</p> <p>The TRNG_TEST.SEL bit field configures the number of the FRO to be tested, the value should be in the range of 0 to 7.</p>				
6 (R/W)	PPROC	<p>Test Post Proc.</p> <p>When the TRNG_TEST.PPROC bit is set, it provides direct access to the post-processor for known-answer tests (writing input data to the TRNG_INPUT[n] registers). While this bit is set, the TRNG can continue to generate entropy in the main LFSR and any buffered random data is preserved, to be loaded into the output registers as soon as this bit is reset to 0 again. The need clock output is forced active while this bit is set.</p> <p>For X9.31 post-processors, it is advisable to re-seed the post-processor after running known-answer tests as the original key and V values are modified to known values.</p> <p>This bit is only present when post-processing is available and can only be set to 1 when the TRNG_CTL.PPROCEN bit =1 and the test run poker (TRNG_TEST.RUNPKR) bit in this register is 0.</p>				
5 (R/W)	RUNPKR	<p>Test Run Poker.</p> <p>When the TRNG_TEST.RUNPKR bit is set, it provides direct access to the inputs of the Monobit, Run and Poker Test circuits (writing input data in chunks of 32 bits to the TRNG_INPUT0 register). While this bit is 1, the TRNG is not allowed to generate entropy but any buffered random data is preserved, to be loaded into the output registers as soon as this bit is reset to 0 again. The TRNG_STAT.NEEDCLK bit is forced active while this bit is 1. The Monobit, Run and Poker Test circuits are reset to their initial states on any change of this bit.</p>				
4 (R/W)	CONTPKR	<p>Continue Poker.</p> <p>When the TRNG_TEST.CONTPKR bit is set, Monobit Test and Poker Test keep running continuously by not resetting the Monobit count (TRNG_MONOBITCNT) and poker counters (TRNG_POKER[n] register) at the end of each 20,000 bits test block. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.</p> <table border="1" data-bbox="620 1690 1521 1787"> <tr> <td>0</td> <td>Do not continue poker test</td> </tr> <tr> <td>1</td> <td>Continue poker test</td> </tr> </table>	0	Do not continue poker test	1	Continue poker test
0	Do not continue poker test					
1	Continue poker test					

Table 36-26: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	NOLFSRFB	No LFSR Feedback. When the TRNG_TEST.NOLFSRFB bit is set, it removes XNOR feedback from the main LFSR, converting it into a normal shift register for the XOR-ed outputs of the FROs (shifting data in on the LSB side). A 1 also forces the LFSR to sample continuously. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.
		0 Keep XNOR feedback
		1 Remove XNOR feedback
2 (R/W)	PATTDDET	Test Pattern Detect. When the TRNG_TEST.PATTDDET bit is set, it repeatedly feeds test pattern (PATTERN) into the error detection circuit of the FRO selected by the test select (TRNG_TEST.SEL) field. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.
		0 Do not repeat feed test pattern
		1 Repeat feed test pattern
1 (R/W)	PATTFRO	Test Pattern FRO. When the TRNG_TEST.PATTFRO bit is set, it repeatedly feeds test pattern (PATTERN) into the delay chain of the FRO selected by the test select (TRNG_TEST.SEL) field by forcing the corresponding FRO enable (FROEN) output LOW. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.
		0 Do not repeat feed test pattern
		1 Repeat feed test pattern
0 (R/W)	ENOUT	Test Enable Out. When the TRNG_TEST.ENOUT bit is set, it enables the tst_fro_clk_out output, connecting to the FRO selected by the test select (TRNG_TEST.SEL) field. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.
		0 Disable tst_fro_clk_out
		1 Enable tst_fro_clk_out

TRNG Post-Process "V" Value Registers

The `TRNG_V[n]` registers are used to load the V value used for post-processing (if available). These registers are write-only. Reads return the values of the other registers mapped at the same addresses.

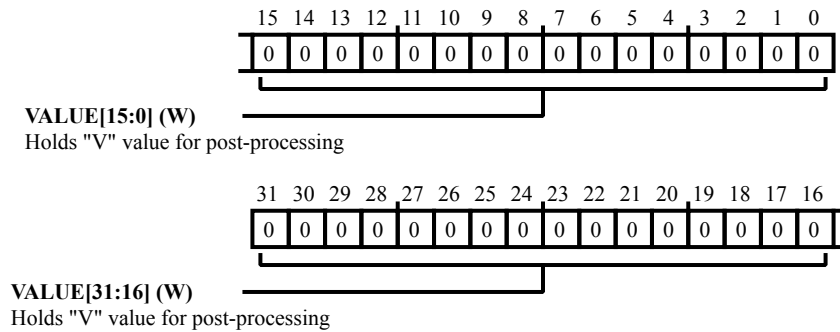


Figure 36-24: TRNG_V[n] Register Diagram

Table 36-27: TRNG_V[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Holds "V" value for post-processing. Bits of the post-processing 'V' value.

37 Thermal Monitoring Unit (TMU)

The TMU provides on-chip temperature measurement which is important in applications that have substantial power consumption. The TMU is integrated into the processor die and digital infrastructure using an MMR-based system access to measure the die temperature variations in real-time.

TMU Features

The TMU supports the following features:

- On-chip temperature sensing
- Programmable over-temperature and under-temperature limits
- Programmable conversion rate
- Averaging feature available
- Temperature gain and offset correction options
- Uses dedicated channel of HADC for conversion
- Programmable blanking and refresh period for TMU conversion

TMU Functional Description

Following sections provide the functional description of TMU.

ADSP-2156x TMU Register List

Thermal monitoring unit

Table 37-1: ADSP-2156x TMU Register List

Name	Description
TMU_ALERT_LIM_HI	Alert High Limit Register
TMU_ALERT_LIM_LO	Alert Low Limit Register
TMU_AVG	Averaging Register

Table 37-1: ADSP-2156x TMU Register List (Continued)

Name	Description
TMU_CNV_BLANK	Temperature conversion blank register
TMU_CTL	TMU Control Register
TMU_FLT_LIM_HI	Fault High Limit Register
TMU_FLT_LIM_LO	Fault Low Limit Register
TMU_GAIN	Gain Value Register
TMU_IMSK	Interrupt Mask Register
TMU_OFFSET	Offset Register
TMU_REFR_CNTR	Temperature Refresh Counter
TMU_STAT	Status Register
TMU_TEMP	Temperature Value Register

ADSP-2156x TMU Interrupt List

Table 37-2: ADSP-2156x TMU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
6	TMU0_FAULT	TMU0 Fault		
7	TMU0_ALERT	TMU0 Fault		

ADSP-2156x TMU Trigger List

Table 37-3: ADSP-2156x TMU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
90	TMU0_FAULT	TMU0 TM0 Fault Event	
91	TMU0_ALERT	TMU0 TM0 Alert Event	

Table 37-4: ADSP-2156x TMU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

TMU Definitions

The following definitions are helpful when using the TMU module.

Thermal Diode

A special type of diode whose electrical properties change with the temperature in a defined way.

DTM

Dynamic thermal management is a set of techniques that adapt the run-time behavior of a processor to achieve the highest performance under thermal constraints.

TMU Block Diagram

The *Thermal Monitoring Unit Block Diagram* shows the main block inside TMU.

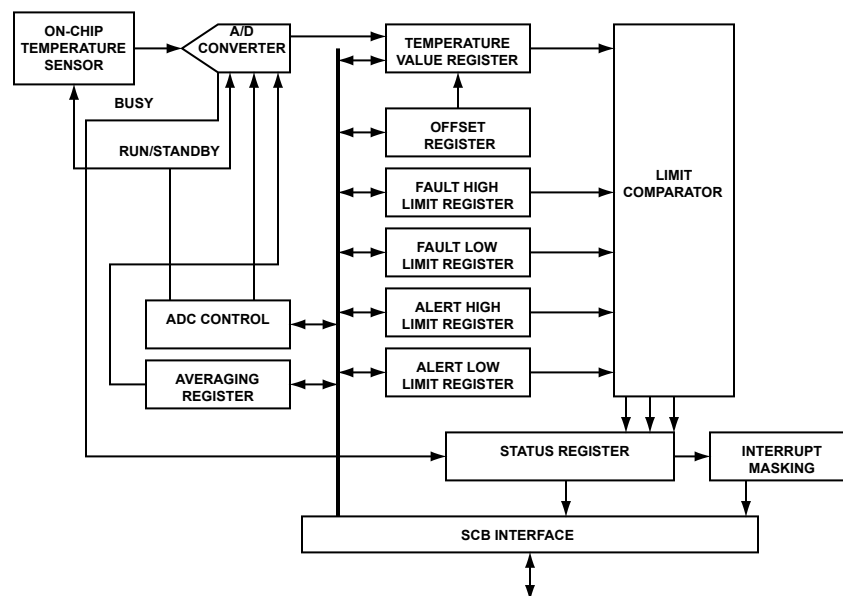


Figure 37-1: Thermal Monitoring Unit Block Diagram

TMU Architectural Concepts

The following sections provide information on the architecture and system integration of the TMU.

System Integration

Dynamic thermal management (DTM) techniques adapt the run-time behavior of a processor to achieve the highest performance under thermal constraints. One of the most important aspects of DTM is to capture the run-time variations in the temperature caused by power consumption variations due to workload changes.

The accuracy of thermal measurements directly affects the efficiency of thermal management as well as the performance of the processor. Temperature estimations lower or higher than the actual temperature may cause late or early activation of DTM techniques.

- Late activation of DTM can result in degraded reliability because the temperature may exceed the designated thresholds.
- Early activation of DTM can have significant impact on performance.

The TMU is an analog thermal sensor that consists of a temperature-sensing diode, a calibrated reference current source, and a current comparator. The sensor placement error is one of the most important sources of inaccuracy in values obtained from thermal sensors.

The TMU module provides the temperature monitoring capability to the chip and implements thermal management in the end system. It provides many features which ensure the minimum load on the software and also minimal or no external components for a flexible temperature monitoring system.

Digital Thermometer

The TMU functions as a digital thermometer with an MMR-based system access. The on-die temperature value is measured and digitized through an A/D converter. The temperature value is stored and updated periodically in an MMR register. The TMU can be configured to generate an interrupt as it crosses the upper temperature limit (`TMU_FLT_LIM_HI` register). A thermal event is also routed to the interrupt port of the sensor. The event is then routed to the SEC, to ensure that core intervention is not required in the event of overheating.

Temperature Sensor Averaging

The temperature sensor averaging feature enhances the accuracy of the temperature measurements. To enable this feature, the `TMU_AVG` bits must be enabled in the control register after power-up. In this mode, the averaging reduces the effect of noise on the temperature result. The temperature is measured each time a conversion is performed. A moving average method is used to determine the result in the temperature value register. The total time to measure a temperature channel is typically 1 ms.

TMU and HADC

The TMU module uses one of the ADC channels to digitize the temperature reading. Therefore, the HADC module is shared by the TMU. If the TMU is not enabled, the HADC operates at its full throughput. TMU can be enabled periodically or permanently. In periodic enable mode, it is enabled once in a specific refresh period. After enabling, the TMU measures the temperature for a specific blanking period. During the blanking period, the TMU uses the ADC for temperature measurement, so that the ADC is not available for data conversion.

The typical value of blanking period is 1×10^6 system clock (SCLK) cycles

The typical value of refresh period is 21×10^6 SCLK cycles

So, for every 21×10^6 cycles, 1×10^6 cycles are used for temperature measurement. Program these values for blanking period and refresh period through the TMU registers.

When TMU is enabled, the HADC module is not available for external input conversion. This state is indicated by the `HADC_STAT.TMUHADC_BUSY` status bit. All the requests for HADC conversion are ignored when `HADC_STAT.TMUHADC_BUSY` bit is high.

In the periodic enable mode, the ongoing HADC conversion is completed activating the TMU to measure temperature as shown. If HADC is in autoscan mode, the TMU is activated after completing the ongoing sequence. If HADC is in fixed-conversion mode, all the fixed conversions are completed and followed by TMU activation.

TMU Event Control

The TMU generates different events depending on the state of the TMU temperature measurement and the different thresholds set in threshold registers. These events are reported in the `TMU_STAT` register as shown below. It can generate an event for each of following conditions.

- The fault high limit is configured in the `TMU_FLT_LIM_HI` register. The interrupt is generated when the temperature value is greater than or equal to this value.
- The alert high limit is configured in the `TMU_ALRT_LIM_HI` register. The interrupt is generated when the temperature value is greater than or equal to this value.
- The fault low limit is configured in the `TMU_FLT_LIM_LO` register. The `TMU_STAT.FLTLO` status bit is set when the temperature value is less than or equal to this value.
- The alert low limit is configured in the `TMU_ALRT_LIM_LO` register. The `TMU_STAT.FLTLO` status bit is set when the temperature value is less than or equal to this value.

Interrupts and status conditions can be masked (disabled) or unmasked (enabled) by setting and clearing bits in the `TMU_IMSK` register.

NOTE: Only `TMU_FLT_LIM_HI` and `TMU_ALRT_LIM_HI` generate an interrupt to core. Other events only result in status change in the `TMU_STAT` register.

Status and Error Signals

When the measured temperature value exceeds the high or low limits that are configured in the `TMU_FLT_LIM_HI/TMU_FLT_LIM_LO` and the `TMU_ALRT_LIM_HI/TMU_ALRT_LIM_LO` registers, corresponding thermal events are triggered.

The fault and alert events are sent to the core through the status register (`TMU_STAT`). Both the `TMU_STAT.FLTHI` and `TMU_STAT.ALRTHI` bits are sticky bits and are cleared by a W1C operation by the core.

The alert and fault registers can be programmed through the MMR bus interface as shown in the TMU block diagram. A write into the `TMU_FLT_LIM_HI/TMU_FLT_LIM_LO` and `TMU_ALRT_LIM_HI/TMU_ALRT_LIM_LO` registers or the `TMU_TEMP` register is not allowed when the events are being triggered.

TMU Programming Guidelines

The following section provides basic programming information for the TMU module.

To get the best performance and accuracy from the TMU, initialize the `TMU_GAIN`, `TMU_OFFSET` and `TMU_AVG` registers before enabling the module. Contact Analog Devices, Inc for the current best values to use.

After these registers are programmed, the rest of the TMU initialization can take place. This includes setting up the fault and alert limits and then enabling the TMU.

Reading Temperature Data

After enabling the TMU, the temperature data can only be read after a finite period of time referred to as startup time.

If the HADC module is used, apply the following mechanism:

1. Initialize the HADC module.
2. Wait for the ADC ready interrupt.
3. Perform TMU conversions.

The minimum time required before reading the first valid data is ~100us for a typical SCLK frequency of 125Mhz and using the default register values.

If the HADC module is not used, complete the following steps

1. Initialize the HADC module (dummy initialization).
2. Wait for the ADC ready interrupt.
3. Perform TMU conversions.

ADSP-2156x TMU Register Descriptions

Thermal monitoring unit (TMU) contains the following registers.

Table 37-5: ADSP-2156x TMU Register List

Name	Description
<code>TMU_ALRT_LIM_HI</code>	Alert High Limit Register
<code>TMU_ALRT_LIM_LO</code>	Alert Low Limit Register
<code>TMU_AVG</code>	Averaging Register
<code>TMU_CNV_BLANK</code>	Temperature conversion blank register
<code>TMU_CTL</code>	TMU Control Register
<code>TMU_FLT_LIM_HI</code>	Fault High Limit Register
<code>TMU_FLT_LIM_LO</code>	Fault Low Limit Register
<code>TMU_GAIN</code>	Gain Value Register
<code>TMU_IMSK</code>	Interrupt Mask Register

Table 37-5: ADSP-2156x TMU Register List (Continued)

Name	Description
TMU_OFFSET	Offset Register
TMU_REFR_CNTR	Temperature Refresh Counter
TMU_STAT	Status Register
TMU_TEMP	Temperature Value Register

Alert High Limit Register

The `TMU_ALERT_LIM_HI` register sets the temperature alert high limit as an integer value. The value is stored in two's complement format. Asserts `TMU_STAT.ALRTHI` if the `TMU_TEMP` value is greater than or equal to `TMU_ALERT_LIM_HI`. The `TMU_ALERT_LIM_HI` value should be programmed for value greater than 8'h3C (60 degC).

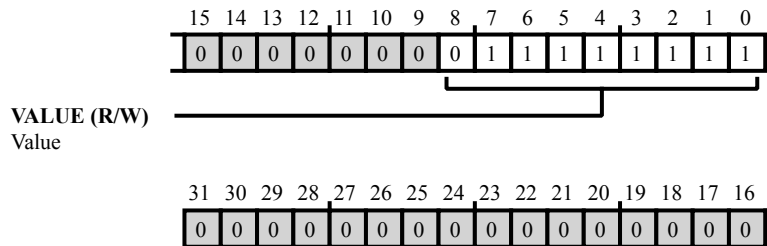


Figure 37-2: `TMU_ALERT_LIM_HI` Register Diagram

Table 37-6: `TMU_ALERT_LIM_HI` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Value. The <code>TMU_ALERT_LIM_HI.VALUE</code> bit field configures the alert temperature high limit as an integer value stored in two's complement format. If the temperature value is greater than or equal to the high-limit the <code>TMU_STAT.ALRTHI</code> bit is set.

Alert Low Limit Register

The `TMU_ALERT_LIM_LO` register configures the alert temperature low limit.

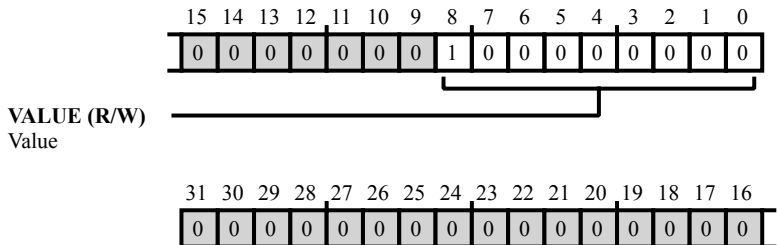


Figure 37-3: `TMU_ALERT_LIM_LO` Register Diagram

Table 37-7: `TMU_ALERT_LIM_LO` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Value. The <code>TMU_ALERT_LIM_LO.VALUE</code> bit field configures the alert temperature low limit as an integer value stored in two's complement format. If the temperature value is less than or equal to the low-limit the <code>TMU_STAT.ALRTLO</code> bit is set.

Averaging Register

The `TMU_AVG` register enables averaging on the TMU.

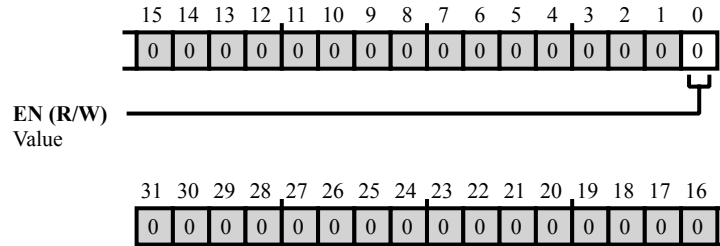


Figure 37-4: `TMU_AVG` Register Diagram

Table 37-8: `TMU_AVG` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	EN	Value. The <code>TMU_AVG.EN</code> bit enables averaging on the TMU. Averaging is done using the formula $(7 \times \text{previous_avg_value} + \text{current_value})/8$. Initially the <code>current_value</code> is taken as <code>previous_avg_value</code> .
		0 No averaging
		1 Enable averaging

Temperature conversion blank register

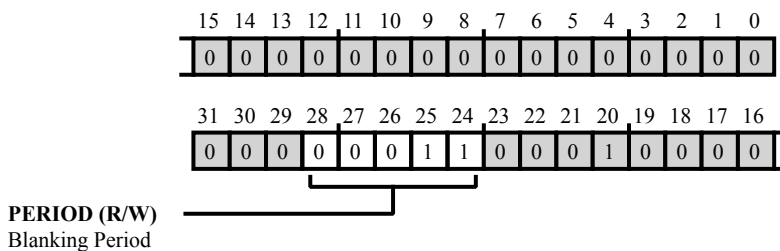


Figure 37-5: TMU_CNVS_BLANK Register Diagram

Table 37-9: TMU_CNVS_BLANK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28:24 (R/W)	PERIOD	Blanking Period. The TMU_CNVS_BLANK.PERIOD specifies the blanking period. TMU uses the HADC for temperature conversion during this period. The conversion time is calculated as: (CNV_BLANK+1)*50k SCLK cycles. So, the minimum conversion time is 50k SCLK cycles. Default conversion time is 20*10 ⁴ SCLK cycles. For this duration, the system cannot use HADC for data conversion and HADC does take input for data-conversion. The status is indicated by TMUHADC_BUSY bit of the HADC block.

TMU Control Register

The `TMU_CTL` register contains bits that allow programs to configure and enable the TMU.

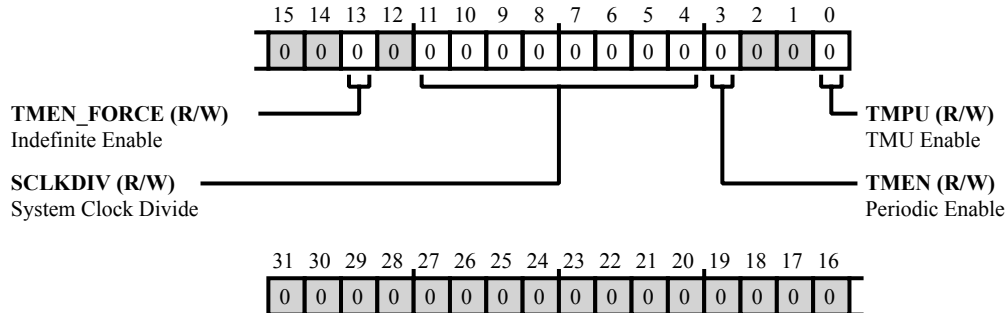


Figure 37-6: TMU_CTL Register Diagram

Table 37-10: TMU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	TMEN_FORCE	Indefinite Enable. Asserting the <code>TMU_CTL.TMEN_FORCE</code> bit enables the TMU indefinitely. HADC is always used by the TMU, so the HADC cannot be used to convert channel inputs.
11:4 (R/W)	SCLKDIV	System Clock Divide. The <code>TMU_CTL.SCLKDIV</code> bit selects the division ratio for the system clock (SCLK). SCLK is divided by $(21+4*SCLKDIV)$. Thus, by default the SCLK is divided by 21.
3 (R/W)	TMEN	Periodic Enable. Asserting the <code>TMU_CTL.TMEN</code> bit enables the TMU periodically. TMU is enabled once in every refresh period defined by the <code>TMU_REFR_CNTR</code> register. And in each refresh period, TMU measures temperature for the blanking period (Tblank). It is defined by CNV_BLANK bit.
		0 Disable TMU
		1 Enable TMU periodically
0 (R/W)	TMPU	TMU Enable. The <code>TMU_CTL.TMPU</code> bit enables the TMU. By default, the module is in power-down mode. The peripheral interface is active even in power-down mode (the TMU registers can be read/written).
		0 Power Down the analog circuitry of TMU
		1 Power Up the analog circuitry of TMU

Fault High Limit Register

The `TMU_FLT_LIM_HI` register sets the temperature fault high limit as an integer value. The value is stored in two's complement format. Asserts `TMU_STAT.FLTHI` if the `TMU_TEMP` value is greater than or equal to `TMU_FLT_LIM_HI`. The `TMU_FLT_LIM_HI` value should be programmed for value greater than 8'h3C (60 degC).

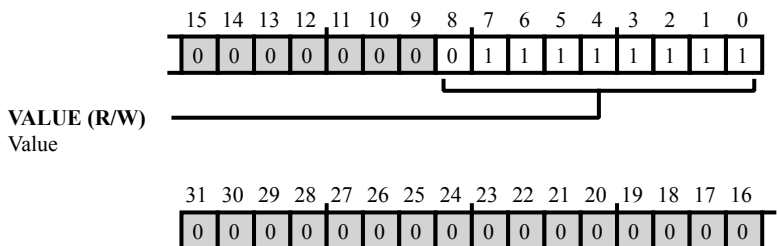


Figure 37-7: TMU_FLT_LIM_HI Register Diagram

Table 37-11: TMU_FLT_LIM_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Value. The <code>TMU_FLT_LIM_HI.VALUE</code> bit field sets the temperature high limit as an integer in two's complement format. If the limit value is greater than or equal to the high-limit the <code>TMU_STAT.FLTHI</code> bit is set.

Fault Low Limit Register

The `TMU_FLT_LIM_LO` register configures the fault temperature low limit.

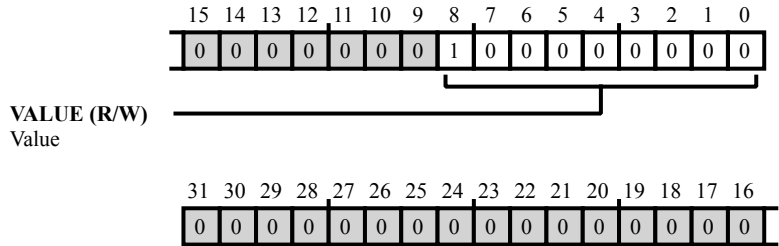


Figure 37-8: `TMU_FLT_LIM_LO` Register Diagram

Table 37-12: `TMU_FLT_LIM_LO` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Value. The <code>TMU_FLT_LIM_LO.VALUE</code> bit field configures the fault temperature low limit as an integer which is stored in two's complement format. If the temperature value is less than or equal to the low-limit, the <code>TMU_STAT.FLTLO</code> bit is set.

Gain Value Register

The `TMU_GAIN` register is used to configure the gain value in two's complement format. This value is used to correct the gain error in the `TMU_TEMP` register.

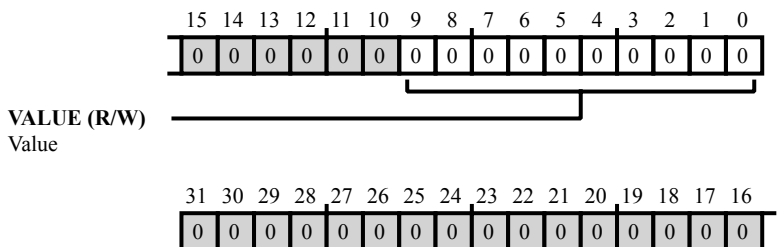


Figure 37-9: `TMU_GAIN` Register Diagram

Table 37-13: `TMU_GAIN` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Value. The <code>TMU_GAIN.VALUE</code> bit field configures the gain value in two's complement format. This value is used to correct the gain error in the <code>TMU_TEMP</code> register.

Interrupt Mask Register

The `TMU_IMSK` register provides bit that are used to mask and unmask the interrupts associated with the TMU module.

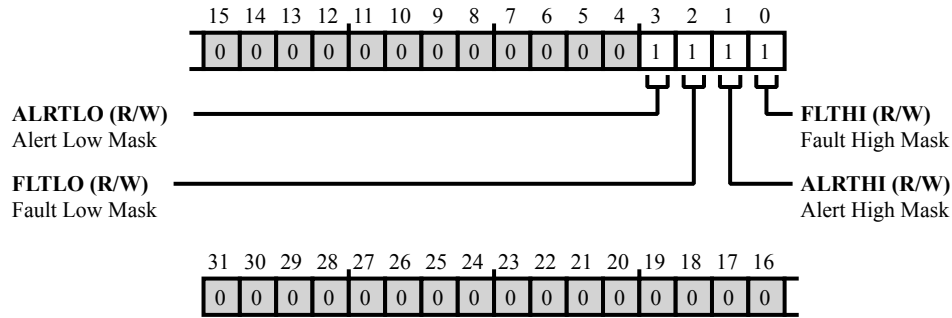


Figure 37-10: `TMU_IMSK` Register Diagram

Table 37-14: `TMU_IMSK` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	ALRTLO	Alert Low Mask. The <code>TMU_IMSK.ALRTLO</code> bit masks or unmasks the alert low status change (<code>TMU_STAT.ALRTLO</code>).
		0 Unmask alert low status change
		1 Mask alert low status change
2 (R/W)	FLTLO	Fault Low Mask. The <code>TMU_IMSK.FLTLO</code> bit masks or unmasks the fault low status change (<code>TMU_STAT.FLTLO</code>).
		0 Unmask fault low status change
		1 Mask fault low status change
1 (R/W)	ALRTHI	Alert High Mask. The <code>TMU_IMSK.ALRTHI</code> bit masks or unmasks the alert high status change (<code>TMU_STAT.ALRTHI</code>).
		0 Unmask alert high status change
		1 Mask alert high status change
0 (R/W)	FLTHI	Fault High Mask. The <code>TMU_IMSK.FLTHI</code> bit masks or unmasks the fault high interrupt (<code>TMU_STAT.FLTHI</code>).
		0 Unmask fault high interrupt
		1 Mask fault high interrupt

Offset Register

The value programmed in the `TMU_OFFSET` register is used to correct the offset error in `TMU_TEMP` register.

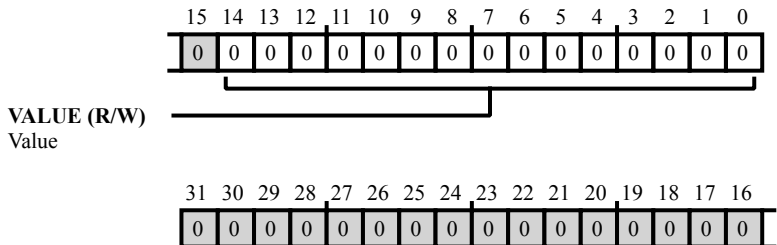


Figure 37-11: `TMU_OFFSET` Register Diagram

Table 37-15: `TMU_OFFSET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Value. The <code>TMU_OFFSET.VALUE</code> bit field provides the offset value which is used to correct the offset error in the <code>TMU_TEMP</code> register. This value is in a two's complement fixed point Q7.7 format i.e 8 MSB's represent integer part in 2's complement format. and remaining 7 LSB's represent decimal part. Offset will be applied after multiplying gain.

Temperature Refresh Counter

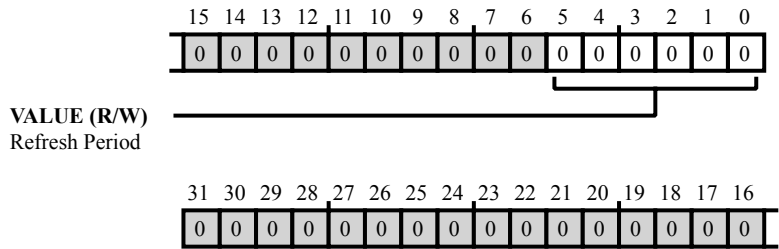


Figure 37-12: TMU_REFR_CNTR Register Diagram

Table 37-16: TMU_REFR_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	Refresh Period. The <code>TMU_REFR_CNTR.VALUE</code> bit field defines the period to refresh the temperature value. Temperature is refreshed for every $(VALUE+1) \cdot (21+4 \cdot SCLKDIV) \cdot 1M$ system-clock cycles. <code>SCLKDIV</code> is defined in <code>TMU_CTL</code> register. So, by default, the period is 21 million system-clock cycles.

Status Register

The `TMU_STAT` register bits indicate when an error or fault is detected.

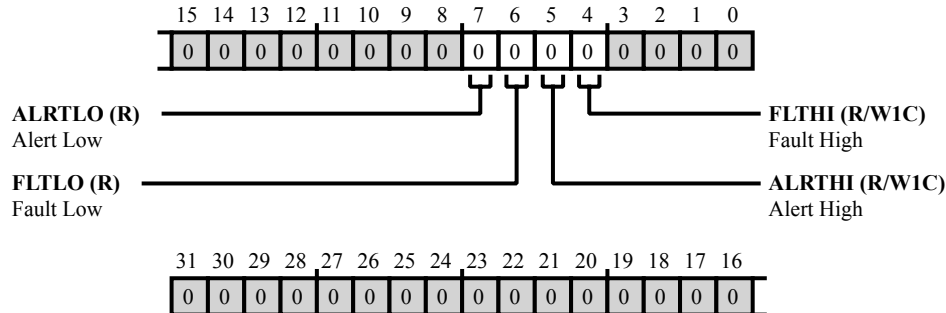


Figure 37-13: `TMU_STAT` Register Diagram

Table 37-17: `TMU_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	ALRTLO	Alert Low. The <code>TMU_STAT.ALRTLO</code> bit is set when the temperature value is less than or equal to the setting in the <code>TMU_ALERT_LIM_LO</code> register. The bit assertion is masked if <code>TMU_IMSK.ALRTLO</code> is HIGH.
6 (R/NW)	FLTLO	Fault Low. The <code>TMU_STAT.FLTLO</code> bit is set when the temperature value is less than or equal to the setting in the <code>TMU_FLT_LIM_LO</code> register. The bit assertion is masked if <code>TMU_IMSK.FLTLO</code> is HIGH.
5 (R/W1C)	ALRTHI	Alert High. The <code>TMU_STAT.ALRTHI</code> bit is set when the temperature value is greater than or equal to the setting in the <code>TMU_ALERT_LIM_HI</code> register. The temperature value register is set to <code>16'h2C00</code> (effective temperature value of 44 degrees C) when a W1C operation is done on this bit. The bit assertion is masked if <code>TMU_IMSK.ALRTHI</code> is HIGH.
4 (R/W1C)	FLTHI	Fault High. The <code>TMU_STAT.FLTHI</code> bit is set when the temperature value is greater than or equal to the setting in the <code>TMU_FLT_LIM_HI</code> register. The temperature value register is set to <code>16'h2C00</code> (effective temperature value of 44) when a W1C is done on this bit. The bit assertion is masked if <code>TMU_IMSK.FLTHI</code> is HIGH.

Temperature Value Register

The `TMU_TEMP` register provides the temperature value from the A/D converter and the status.

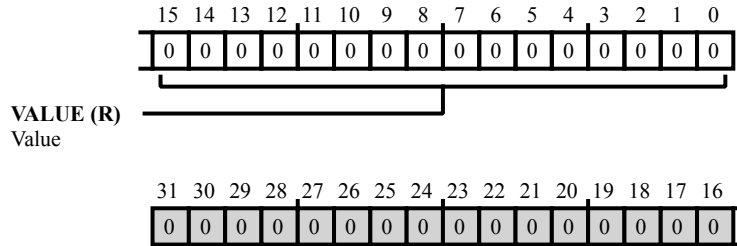


Figure 37-14: `TMU_TEMP` Register Diagram

Table 37-18: `TMU_TEMP` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Value. The <code>TMU_TEMP.VALUE</code> bit field is the temperature value from A/D converter. This value is stored in two's complement fixed point Q7.8 format where the 8 MSB's represent the integer part in 2's complement, and the remaining 8 LSB's represent the decimal part.

38 FIR Accelerator (FIR)

Finite Impulse Response (FIR) filters are frequently used in digital signal processing applications. The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core. FIR filters are used in a wide array of applications including multi-rate processing with an interpolator or decimator.

Features

This hardware module can perform FIR filters without core intervention. This feature gives programs freedom to use the core to implement complex algorithms, effectively adding more bandwidth to the processor.

The FIR supports the following features:

- Fixed-point and 32-bit IEEE floating-point format
- Four SHARC+ core compatible MAC units that operate in parallel
- Rounding modes compatible with SHARC+ core MACs
- Single rate or multi-rate window processing
- Programmable rates with decimation or interpolation mode
- Up to 32 filter channels available in TDM in legacy mode
- Burst transfers on data or coefficient loads

NOTE: The FIR accelerator module has a local memory that the core cannot access during regular operation. Unlike previous SHARC processors, the FIR accelerator modules each have access to the system memory (on-chip or off-chip).

Also, unlike the previous SHARC processors, where only one of the FIR or IIR accelerators can be used at a time, the SHARC+ processor can use both accelerators simultaneously.

Clocking

The FIR accelerator runs at the speed of the core clock frequency (CCLK).

Functional Description

The *FIR Block Diagram* shows the 1024-TAP FIR hardware accelerator. The accelerator consists of a 1024 word coefficient memory, a 1024 deep delay line for data, and four MAC units. The accelerator runs at CCLK frequency.

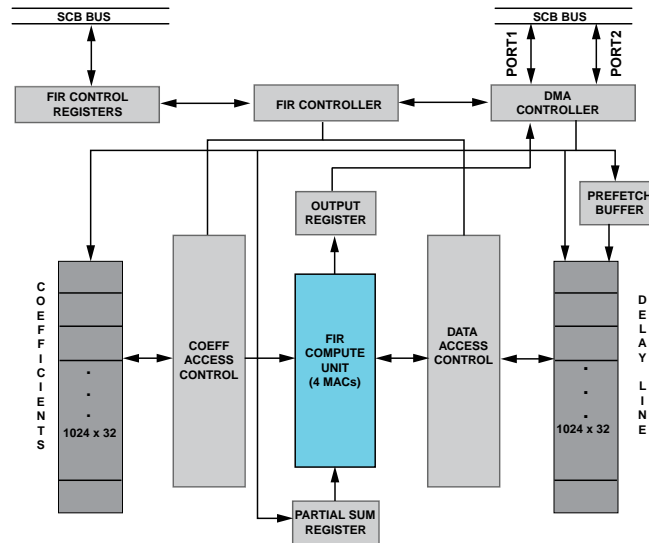


Figure 38-1: FIR Block Diagram

The FIR accelerator has the following logical sub blocks:

1. A datapath unit that consists of:
 - A 1024 deep coefficient memory
 - A 1024 deep delay line for the data
 - Four 32-bit floating-point and fixed-point multiplier and adder units
 - One 32-bit prefetch buffer to operate in a pipelined fashion
 - One 32-bit buffer to hold the previous partial sum
 - One 32-bit buffer to hold the output
2. Configuration registers for the number of TAPs, number of channels, filter enable, interrupt control, DMA enable, up sample or down sample control, and ratios.
3. Core access interface for writing to the DMA and filter configuration registers and reading the status register.
4. DMA bus interface for transferring data and coefficients to and from the accelerator.
5. DMA configuration registers including chain pointer, input, output, and coefficient registers.

The accelerator block is integrated with the SHARC core. For more information about this, see [Figure 41-4 SHARC SCB with FIR/IIR Accelerator](#).

ADSP-2156x FIR Register List

The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core.

Table 38-1: ADSP-2156x FIR Register List

Name	Description
FIR_CHNPTR	FIR Chain Pointer Register
FIR_COEFCNT	FIR Coefficient Count Register
FIR_COEFIDX	FIR Coefficient Index Register
FIR_COEFMOD	FIR Coefficient Modifier Register
FIR_CTL1	FIR Global Control Register
FIR_CTL2	FIR Channel Control Register
FIR_DBG_ADDR	Debug Address Register
FIR_DBG_CTL	FIR Debug Control Register
FIR_DBG_RDDAT	FIR Debug Data Read Register
FIR_DBG_WRDAT	FIR Debug Data Write Register
FIR_DMASTAT	FIR DMA Status Register
FIR_INBASE	FIR Input Data Base Register
FIR_INCNT	FIR Input Data Count Register
FIR_INIDX	FIR Input Data Index Register
FIR_INMOD	FIR Input Data Modifier Register
FIR_MACSTAT	FIR MAC Status Register
FIR_OUTBASE	FIR Output Data Base Register
FIR_OUTCNT	FIR Output Data Count Register
FIR_OUTIDX	FIR Output Data Index Register
FIR_OUTMOD	FIR Output Data Modifier Register
FIR_SCTL1	software control register 1
FIR_SCTL2	software control register 2
FIR_SGCTL	Secondary global control register

ADSP-2156x FIR Interrupt List

Table 38-2: ADSP-2156x FIR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
118	FIR0_DMA	FIR0 DMA	Edge	
119	FIR0_STAT	FIR0 Status	Edge	

ADSP-2156x FIR Trigger List

Table 38-3: ADSP-2156x FIR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
39	FIR0_DMA	FIR0 DMA	Edge
103	FIR0_TRGO	FIR0 FIR Trigger Output	Edge

Table 38-4: ADSP-2156x FIR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
116	FIR0_TRGI	FIR0 FIR Trigger input	Pulse

Compute Block

The MAC unit, shown in the *FIR MAC Unit* figure, has four multiply accumulators. The accumulators operate simultaneously on a single filter as described below.

- The MAC unit operates on the data and coefficient fetched from the data and coefficient RAMs
- Each MAC can perform 32-bit floating-point or 32-bit fixed-point MAC operations
- Floating-point format is IEEE-compliant
- Multiply and accumulation operation (addition) are pipelined
- A 32-bit floating-point MAC operation generates 32-bit multiply results
- A 32-bit fixed-point operation generates 80-bit results (64-bit result + 16 guard bits)

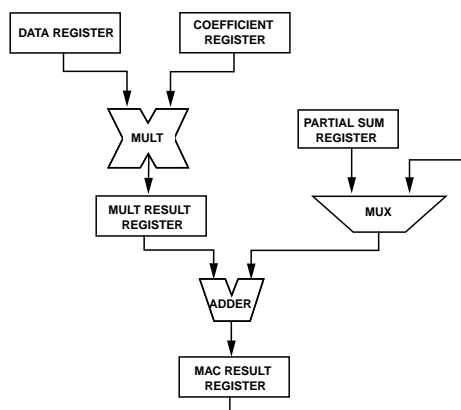


Figure 38-2: FIR MAC Unit

Partial Sum Register

The partial sum register is useful for [Floating-point Multi-Iteration](#) mode. For a particular channel, the intermediate MAC result is written to the output buffer of the system (on-chip or off-chip) memory. If the same channel is requested again, the partial result register is updated with the intermediate MAC result through DMA from output buffer of the system memory. The result is added to the current MAC result after each iteration. This process repeats until all iterations complete (the entire soft filter length is processed).

Delay Line Memory

The accelerator has a 1024 TAP delay line to hold the data locally. The DMA controller fetches the data from system memory and loads it into the delay line. Four read accesses can be made to the delay line simultaneously.

Coefficient Memory

The accelerator has a 1024 deep coefficient memory to store the coefficients. The DMA controller loads the coefficients from system memory into coefficient memory. Four coefficients can be fetched from the coefficient memory simultaneously. If the soft filter length is more than 1024, processing happens in multi-iteration mode.

Prefetch Data Buffer

The prefetch data buffer enables pipeline operation. One data sample is prefetched when the compute unit is operating on the delay line corresponding to the current sample. The data prefetched in this buffer is later used to update the delay line for the next sample. This operation happens in parallel again when the compute unit is not accessing the delay line. In other words, it happens when the compute unit is adding the output from the four MACs and the partial sum register.

Table 38-5: Pipeline Operation for Window Size = 1

Cycles	1	2	3	4	5	6
<i>Output DMA</i>			N	N1	N2	N3
<i>Compute</i>		N	N1	N2	N3	

Table 38-5: Pipeline Operation for Window Size = 1 (Continued)

Cycles	1	2	3	4	5	6
<i>Input DMA</i>	N	prefetch N1	prefetch N2	prefetch N3		

Processing Output

The accelerator uses all four MACs simultaneously to calculate one output sample as shown in the *Multi-Iteration Filtering Flow* figure and the following procedure.

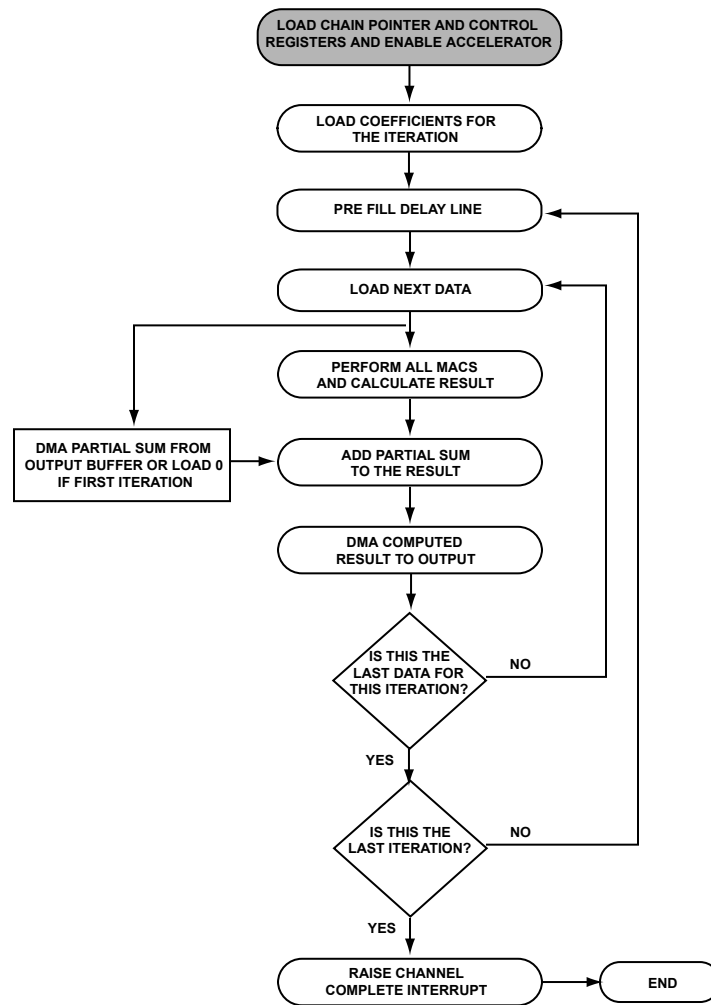


Figure 38-3: Multi-Iteration Filtering Flow

1. The accelerator fetches four input data from the delay line and four corresponding coefficients from the coefficient memory and feeds them to the MAC units for multiply and accumulation operations.
2. The accelerator repeats the procedure with the next four input data and coefficients until all the TAPs complete. For example, this procedure happens $N/4$ times for an N TAP filter.

3. When all the TAPs are complete, the accelerator adds the four MAC outputs together to the previous partial sum (if any) to calculate the final result.
4. Finally, that output sample is stored back in system memory.

System Memory Storage

The following sections describe the storage format for the accelerator.

CAUTION: Store all data in memory aligned to the word address boundaries. Any other programmed addresses do not flag an error.

Coefficients and Input Buffer Storage

For any N TAP filter with coefficients:

```
C[i] i = 0,1,
...
N - 1
```

store the coefficients in system memory buffer in the order:

```
C[N - 1], C[N - 2]
...
C[1], C[0]
```

and CI should point to $C(N - 1)$.

Single Rate Input Filtering

The total size of the input buffer must at least be equal to $N - 1 + W$. If the input buffer that needs to be processed is:

```
x[n], x[n+1], x[n+2]
...
x[n+W-1]
```

store it in the memory as

```
x[n-(N-1)], x[n-(N-2)]
...
x[n-1], x[n], x[n+1]
...
x[n+W-1]
```

and the `FIR_INIDX` register value should point to $x[n - (N - 1)]$

Decimation

Assuming M = decimation ratio, the total size of the input buffer should at least be equal to $N - 1 + W \times M$. If the input buffer that needs to be processed is:

```
x[n], x[n+1], x[n+2] . . . . x[n+W×M-1],
```

store it in the memory as:

```
x[n-(N-1)], x[n-(N-2)]...x[n-1],
x[n], x[n+1]...x[n+WxM-1]
```

and the `FIR_INIDX` register value should point to `x[n-(N-1)]`.

Interpolation

Assuming L = interpolation ratio, the total size of the input buffer should be at least equal to:

$\text{Ceil}((N-1)/L) + W/L$.

If the input buffer that needs to be processed is:

```
x[n], x[n+1], x[n+2]...x[n+W/L-1],
and K = Ceil((N-1)/L)
```

store it in the memory as:

```
x[n-k], x[n-(K-1)], x[n-(K-2)]...x[n-1],
x[n], x[n+1]... x[n+W/L-1]
```

and the `FIR_INIDX` register value should point to `x[n-K]`.

Operating Modes

The FIR core performs a sum-of-products operation to compute the convolution sum. It supports single-rate, decimation, and interpolation functions.

Single Rate Processing

In a single-rate filter, the output result rate is equal to the input sample rate. The filter output $Y(n)$ is computed according to following equation where N is the number of filter coefficients: $c(k)$ $k = 0$ to $N - 1$ are the filter coefficients and $x(n)$ represents the input time-series.

$$Y(n) = \sum_{k=0}^{N-1} c(k) \cdot x(n-k)$$

Figure 38-4: Filter Output Calculation

Single Iteration

Results are computed in a single iteration when the soft filter length is less than or equal to 1024.

Floating-point Multi-Iteration

Results are computed in multiple iterations when the soft filter length is greater than 1024 (for example, 2048 TAPs on a 1024 hard filter length). In this mode, the accelerator implements two iterations of 1024 TAPs.

NOTE: If the soft filter length is not a multiple of the hard filter length, the accelerator iterates until the soft filter length is satisfied.

Example: 550 taps on a 256 tap filter. In this example, the FIR accelerator implements two iterations of 256 taps and one iteration of 38 taps.

NOTE: Multi-iteration mode is not supported in fixed-point format.

Window Processing

In window-based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

To select sample-based processing mode, configure the window size to 1. In this mode, one sample from a particular channel is processed through all the taps of that channel and the final output sample is calculated.

Multi-Rate Processing

Multi-rate filters change the sampling rate of a signal—they convert the input samples of a signal to a different set of data that represents the same signal sampled at a different rate.

Decimation

A decimation filter provides a single output result for every M input samples, where M is the decimation ratio. The output rate is $1/M$ 'th of the input rate. The filter implementation exploits the low output sample rate by not starting a computation until a new set of M input samples is available.

In this mode, after low-pass filtering (for anti-aliasing), FIR logic discards the ratio $- 1$ samples of output data. For performance optimization, FIR logic skips the computation of output samples, which are discarded.

The input buffer size for decimation filters is $N - 1 + (W \times M)$ where:

- N is the number of taps
- W is the window size
- M is the decimation ratio

The window size (`FIR_CTL2.WINDOW` bits) must be programmed with the number of output samples.

To start this mode, programs set the `FIR_CTL2.RATIO` and `FIR_CTL2.UPSAMP` bits (along with normal filter setting). Also, the `FIR_CTL2.TAPLEN` bit field value should be greater than or equal to the `FIR_CTL2.RATIO` bit field value for the decimation filter.

Interpolation

An interpolation filter provides L output results for each new input sample, where L is the interpolation ratio. The output rate is L times the input rate.

In this mode, according to the ratio specified in configuration register, FIR logic inserts $L - 1$ zeros between any two input samples (upsampling). It then performs the interpolation (through the FIR filter).

Both upsampling and downsampling do not support multi-iteration mode. Therefore, the filtering operation only happens on up to 1024 TAPs and the ratio of up and downsampling can only be an integer value.

In an interpolation filter, FIR logic inserts $L - 1$ zeros between each sample. The program has to make sure that these zeros fully shift out of the delay line before moving on to the next channel. This operation puts a restriction on window size in terms of L – *the sample ratio* as showing in the expression:

$WINDOWSIZE = n \times SAMPLERATIO$, where n is the number of input samples.

The input buffer size is smallest integer greater than or equal to $(N - 1 + W)/L$ for interpolation filters where:

- N is the number of taps
- W is the window size
- L is the interpolation ratio

To start the mode, programs configure the `FIR_CTL2 .RATIO` and `FIR_CTL2 .UPSAMP` bits (along with filter settings).

Floating-Point Data Format

The FIR accelerator treats data and coefficients in 32-bit floating-point format as the default functional mode.

Fixed-Point Data Format

In fixed-point mode, the 32-bit input data or coefficient is treated as fixed point. A 32-bit fixed-point MAC operation generates an 80-bit result. Fixed-point data or coefficients can be unsigned integer, unsigned fractional and signed integer.

NOTE: In fixed-point mode, the entire 80-bit result register is always written back in bursts of 3×32 bits. The first word is the LSW, the second word is the MSW, and the third word is a 16-bit overflow. The remaining 16 bits are padded with zeros. Therefore, for fixed-point mode: $WINDOWSIZE = WINDOWSIZE \times 3$.

If the signed fractional format is used, the output must be scaled by 2. The MAC does not right shift to remove the redundant sign bit. A final routine must decimate the output buffer to the desired samples.

Multi-iteration mode is not supported in this format. Therefore, the maximum TAP length is 1024.

Auto Configuration Mode (ACM)

The accelerator can be operated in legacy mode or Auto Configuration Mode (ACM), controlled by the `FIR_CTL1 .ACM` bit. The default functional mode is legacy mode. The accelerator mode can only be changed when the accelerator is disabled.

The ACM provides the following additional features:

- Halt and Queuing

Core may pause the current TCB chain being processed by setting the `FIR_CTL1.HALT` bit. The accelerator acknowledges the core by setting the `FIR_DMASTAT.HALT_STAT` bit. The core can appropriately take action to submit/insert new TCB's. After the core takes action, the accelerator processing can be resumed by clearing the `FIR_CTL1.HALT` bit. Before halting the accelerator, if the initial TCB chain is processed, the accelerator comes to idle state. In this case, the accelerator has to be disabled and enabled by toggling the `FIR_CTL1.EN` bit and the `FIR_CTL1.HALT` bit has to be cleared to resume processing.

- No Channel Number Limitation

Unlike in legacy mode, there is no fixed channel number limitation and, therefore, the accelerator ignores the value programmed in the `FIR_CTL1.CH` field. The application can queue unlimited number of channels/TCBs dynamically and the accelerator keeps processing the TCBs until the chain pointer becomes null.

- Selective Interrupt

Core can enable/mask interrupt generation for each channel using the `FIR_CTL2.IMASK` bit. If the bit is cleared, an interrupt is generated after completion of the channel.

- Selective Master/Slave Trigger

Core can enable/mask trigger generation by the accelerator after the end of processing of each channel using the `FIR_CTL2.TMASK` bit. The accelerator can also wait for a trigger after loading the TCB and coefficients and before processing a channel for which the `FIR_CTL2.TWAIT` bit is set.

In addition to the above features, there are three additional fields as part of the TCB `FIR_SCTL1`, `FIR_SCTL2`, and `FIR_SGCTL`. The `FIR_SGCTL` can be used to change the `FIR_CTL1` parameters such as rounding mode and fixed point mode for each channel. The `FIR_SCTL1` and `FIR_SCTL2` can be used as general-purpose registers.

Data Transfer

The FIR filter works exclusively through DMA.

Chain Assignment

The structure of a TCB is conceptually the same as a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB can point to itself to continuously re-run the same DMA. The FIR accelerator reads each word of the TCB and loads it into the corresponding register. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

The FIR accelerator DMA supports circular buffer chained DMA. The FIR accelerator does not support circular buffering for the coefficient buffer.

Table 38-6: TCBs for Chained DMA in Legacy Mode

Address	Register
TCB	<code>FIR_CHNPTR</code>

Table 38-6: TCBs for Chained DMA in Legacy Mode (Continued)

Address	Register
TCB + 0x1	FIR_COEFCNT
TCB + 0x2	FIR_COEFMOD
TCB + 0x3	FIR_COEFIDX
TCB + 0x4	FIR_OUTBASE
TCB + 0x5	FIR_OUTCNT
TCB + 0x6	FIR_OUTMOD
TCB + 0x7	FIR_OUTIDX
TCB + 0x8	FIR_INBASE
TCB + 0x9	FIR_INCNT
TCB + 0xA	FIR_INMOD
TCB + 0xB	FIR_INIDX
TCB + 0xC	FIR_CTL2

Table 38-7: TCBs for Chained DMA in Auto Configuration Mode

Address	Register
TCB	FIR_CHNPTR
TCB + 0x1	FIR_SCTL1
TCB + 0x2	FIR_SCTL2
TCB + 0x3	FIR_SGCTL
TCB + 0x4	FIR_COEFCNT
TCB + 0x5	FIR_COEFMOD
TCB + 0x6	FIR_COEFIDX
TCB + 0x7	FIR_OUTBASE
TCB + 0x8	FIR_OUTCNT
TCB + 0x9	FIR_OUTMOD
TCB + 0xA	FIR_OUTIDX
TCB + 0xB	FIR_INBASE
TCB + 0xC	FIR_INCNT
TCB + 0xD	FIR_INMOD
TCB + 0xE	FIR_INIDX
TCB + 0xF	FIR_CTL2

The `FIR_COEFCNT` register is loaded with the values in the `FIR_COEFCNT` TCB field and is decremented from that value onwards. However, coefficient loading continues until the number of coefficients, equal to the tap length, are read. This condition is true even if the `FIR_COEFCNT` register reaches zero as in the case of a tap length = 10, and the `FIR_COEFCNT` field in the TCB is initialized to 0. The value in the `FIR_COEFCNT` register is -10 after all coefficients are loaded.

NOTE: Initialize `FIR_CHNPTR` to TCB + 12 in legacy mode and TCB + 15 in ACM.

DMA Access

The FIR accelerator has two DMA channels (accelerator input and output) to connect to the system memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Burst Access Support

Burst access enhances the throughput of the DMA channel and reduces the overall load on the system fabric. Burst support is provided for data and coefficient loads on the DMA channel. The FIR module supports burst transfers of size SINGLE, INCR4 and INCR8.

An additional bit is provided in the `FIR_CTL1` register for enabling the burst feature. Also, burst transfers are always of size SINGLE when the modifier is other than 1. In case of burst transfers around the circular buffer boundary, the design ensures that the burst does not cross the buffer boundary.

Accelerator TCB

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the internal memory of the processor. In chained DMA operations, the accelerator automatically initializes and starts another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One TCB must be configured for each channel. It contains the following:

- A control register value to configure the filter parameters (such as filter tap length, window size, sample rate conversion settings) for each channel. In ACM, additional parameter such as interrupt mask, trigger mask, and trigger wait are also available.
- Software control register values in ACM for each channel.
- Secondary control register value to configure rounding mode, fixed point mode, and two's complement for each channel.
- DMA parameter register values for the input data (delay line).
- DMA parameter register values for coefficient load.
- DMA parameter register values for output data.

Intermediate results in multi-iteration mode are saved in the output buffer.

As shown in the *Circular Buffer Addressing* figure, the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the appropriate values to the `FIR_INIDX` and `FIR_OUTIDX` bit fields of the TCB in memory. Then, data processing can begin from where it left off during the next time slot of that channel.

The write-back value for input buffer is:

- `FIR_INIDX + W` for single rate filtering
- `FIR_INIDX + W × M` for decimation ($M = \text{decimation ratio}$)
- `FIR_INIDX + W/L` for interpolation ($L = \text{interpolation ratio}$)
- The write-back value for output buffer in floating point mode is: `FIR_OUTIDX + W`
- The write-back value for output buffer in fixed-point mode is: `FIR_OUTIDX + 3 × W`

NOTE: The `FIR_CTL2` register is part of the FIR TCB. This configuration allows programming individual FIR channels with different control attributes.

The above index updates are valid only for legacy mode. In ACM, the `FIR_INIDX` and `FIR_OUTIDX` bit fields of the TCB in memory are updated to `0x00000000` and `0xFFFFFFFF` by the accelerator after processing a window of data.

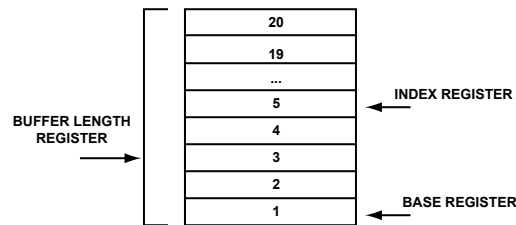


Figure 38-5: Circular Buffer Addressing

Programming Model

The following sections provide general programming information for the FIR accelerator.

Legacy Mode

The *Wait for Core Intervention ≥ Idle (if auto channel iterate bit = 0)* figure shows the diagram for multichannel filtering. Multiple channels are processed in a time division multiplexed (TDM) format. After completing all the channels, the accelerator can either repeat the slots or wait for core intervention.

For multichannel filtering, use the following steps.

1. Program the number of channels using the `FIR_CTL1.CH` bits.
2. Configure the TCBs in system memory with one channel's TCB pointing to the next channel's TCB.

- Write the first TCB value into the `FIR_CHNPTR` register and enable the accelerator.

The accelerator fetches the first channel's TCB and, using it as pointer, prefills the delay line and coefficient memory and loads the `FIR_CTL2` register to configure the filter parameters corresponding to that channel.

The accelerator then calculates output samples corresponding to one window and stores the data back in internal memory.

At the end of the window the accelerator updates the `FIR_INIDX` and `FIR_OUTIDX` registers in the TCB of system memory and moves to the next channel.

When all the channels are finished and the auto channel iterate bit (`FIR_CTL1.CAI`) = 1, the accelerator processes the first channel again and iterates through the channels. If the `FIR_CTL1.CAI` bit = 0, the accelerator waits for core intervention.

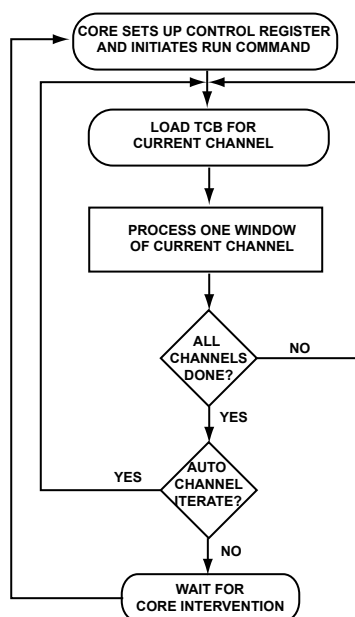


Figure 38-6: Wait for Core Intervention \geq Idle (if auto channel iterate bit = 0)

NOTE: All the addresses programmed in the TCB correspond to 32-bit address boundaries and should not contain the lower 2 bits (assumed as zeros).

Auto Configuration Mode (ACM)

The figure shows multi-channel filtering in ACM. Multiple channels are processed in a time multiplexed format (TDM).

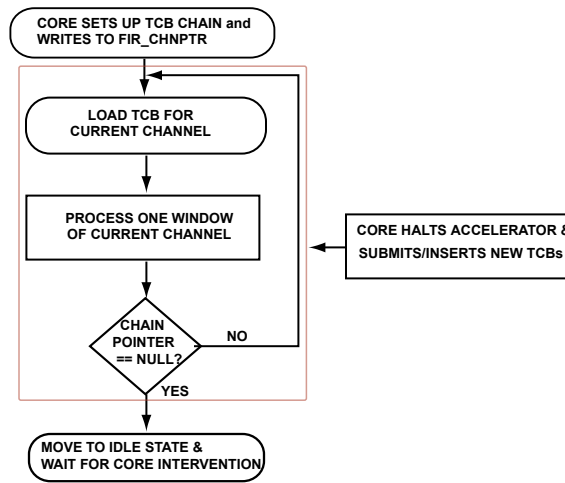


Figure 38-7: Multi-Channel Filtering in Auto Configuration Mode

For multi-channel filtering:

1. Configure the TCBs in system memory with one channel's TCB pointing to the next channel's TCB. There is no limit on the number of channels to be configured.
2. Write the first TCB value to the `FIR_CHNPTR` register and enable the accelerator.

The accelerator fetches the first channel's TCB. Using it as pointer, it prefills the delay line and coefficient memory and loads the `FIR_CTL2` and `FIR_SGCTL` registers to configure the filter parameters corresponding to that channel.

If the `FIR_CTL2.TWAIT` bit is set, the accelerator waits for an input trigger to start the window processing for the channel.

The accelerator then calculates output samples corresponding to one window and stores the data back in internal memory. If the `FIR_CTL2.TMASK` bit is cleared the accelerator sends an output trigger after completion of processing of the channel.

If the `FIR_CTL2.IMASK` bit is cleared, the accelerator interrupts the core after completion of processing of a particular channel.

At the end of the window, the accelerator updates the `FIR_INIDX` and `FIR_OUTIDX` registers to `0x00000000` and `0xFFFFFFFF` in the TCB of system memory and moves to the next channel.

3. At any instant, as required, the core halts the accelerator. It sets the `FIR_CTL1.HALT` bit and appropriately takes action to submit or insert new TCBs and clears the `FIR_CTL1.HALT` bit to resume channel processing.

If the `FIR_CHNPTR` register is zero (last channel is being processed or channel processing is complete) after halting the accelerator, the `FIR_CHNPTR` register is appropriately written before resuming the accelerator channel processing.

If the accelerator is idle after halt, `FIR_CTL1.EN` bit is toggled to disable and re-enable the accelerator and the `FIR_CTL1.HALT` bit is cleared and the accelerator resumes channel processing.

4. The accelerator continues processing until all the channels are complete. Repeat *Step 3* (if required) to submit/insert new channels.

NOTE: Channel Auto Iterate (`FIR_CTL1.CAI`) is not supported in ACM.

Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Clear the `FIR_CTL1.DMAEN` bit.
2. Set the `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.MEM`, and `FIR_DBG_CTL.HLD` bits.
3. Set the `FIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write the start address to the `FIR_DBG_ADDR` register.

NOTE: If the bit 11 in the `FIR_DBG_ADDR` register is set, coefficient memory is selected.

5. Write data to the `FIR_DBG_WRDAT` register.

Read from Local Memory

1. Clear the `FIR_CTL1.DMAEN` bit.
2. Set the `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.MEM`, and `FIR_DBG_CTL.HLD` bits.
3. Set the `FIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write the start address to the `FIR_DBG_ADDR` register.

NOTE: If bit 11 in the `FIR_DBG_ADDR` register is set, coefficient memory is selected.

5. Read data from the `FIR_DBG_RDDAT` register.

Single-Step Mode

Single-step mode can be used for debug purposes. An extra debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the `FIR_DBG_CTL.HLD` bit and enable `FIR_DBG_CTL.EN` and `FIR_DBG_CTL.RUN` bits.
3. Program the FIR module according to the application.
4. In single-step mode, each iteration is updated in the emulator session.

Computing FIR Output, Tap Length Greater than 4096

With little core intervention, the FIR accelerator can also be used to calculate output for a tap length greater than 4096 taps. The section demonstrates the calculation with an example of 8192 taps.

1. Divide the transfer function of an 8192 FIR filter into two 4096 FIR filters:

$$H(Z) = b_0 + b_1Z^{-1} + b_2Z^{-2} + \dots b_{4095}Z^{-4095} + b_{4096}Z^{-4096} + b_{4097}Z^{-4097} + \dots b_{8191}Z^{-8191}$$

$$= b_0 + b_1Z^{-1} + b_2Z^{-2} + \dots b_{4095}Z^{-4095} + Z^{-4096}(b_{4096} + b_{4097} + \dots b_{8191}Z^{-4096})$$

2. Divide the filter coefficients of an 8192 tap filter among two 4096 tap FIR filters:

Filter 1

Coefficients = $b_0, b_1, b_2, \dots, b_{4095}$

Input data = $x[n], x[n-1], \dots, x[n-4095]$

Filter 2

Coefficients = $b_{4096}, b_{4097}, \dots, b_{8191}$

Input data = $x[n-4096], x[n-4097], \dots, x[n-8191]$

The accelerator can be used in two-channel mode where channel 1 operates on $x[n] \dots x[n-4095]$ input data with the filter coefficients of filter 1 and channel 2 operates on $x[n-4096] \dots x[n-8191]$ with the filter coefficients of filter 2.

Once both the channels are processed, add the partial sum output of both the channels to get the final output. Implement this approach (tap length = TAPS = 8192, window size = WINDOW) using the following programming steps.

1. Create a circular input data buffer in system memory (IBUF). The buffer must be large enough to avoid overwriting data before the accelerator processes it. Ideally, the input buffer size for a channel is TAPS + WINDOW - 1.
2. Create a coefficient buffer of size TAPS (8192) (CBUF).
3. Create one output buffer of size WINDOW (OBUF) and another temporary output buffer (OBUF1) to store the partial sum.
4. Create two TCBs in system memory with first TCB chained to the second and second one chained to the first in circular manner.
 - a. The `FIR_COEFIDX` bit field of the first TCB should point to the start address of the coefficient buffer (CBUF) and that of the second TCB should point to 4096 offset from the start of the coefficient buffer (CBUF + 4096).
 - b. The `FIR_OUTBASE` and `FIR_OUTIDX` bit field of the first TCB should point to the start address of OBUF and that of the second TCB should point to the start address of OBUF1.
 - c. The `FIR_INIDX` bit field of the first TCB should point to the start address of IBUF and that of the second TCB should point to 4096 offset from the start address of IBUF.
 - d. Configure the `FIR_CTL2` bit field of both the TCB for tap length = TAP/2 = 4096 and window size = WINDOW.

5. Initialize the `FIR_CHNPTR` register pointing to the first TCB.
6. Program the `FIR_CTL1` register to initiate the accelerator processing now by setting the `FIR_CTL1.EN` and `FIR_CTL1.DMAEN` bits and the number of channels configured as 2.
7. Wait for the FIR all channel done interrupt (`FIR_DMASTAT.ACDONE`) to occur. Inside the ISR, add the partial sum results using the core from both the output buffers (`OBUF` and `OBUF1`) to get the final output. To save memory, replace the contents of the buffer `OBUF` with the final output result.

Debug Features

The following sections provide information for debugging the FIR accelerator.

Local Memory Access

The contents of FIR delay line and coefficient memories are made observable for debug by setting the `FIR_DBG_CTL.EN/FIR_DBG_CTL.MEM` and `FIR_DBG_CTL.HLD` bits. The debug address register (`FIR_DBG_ADDR`) and two data registers are provided for debug operations. Bit 11 of the `FIR_DBG_ADDR` register selects coefficient memory when set (=1) and selects delay line memory when cleared (=0).

In the debug mode, the read data register (`FIR_DBG_RDDAT`) returns the contents of the memory location pointed to by the address register. Data can be written into any memory location using `FIR_DBG_WRDAT` register writes. If the address auto-increment bit (`FIR_DBG_CTL.ADRINC`) is set, the address register auto-increments on `FIR_DBG_WRDAT` writes and `FIR_DBG_RDDAT` reads. During auto-increment the `FIR_DBG_ADDR` register cannot cross the data memory or coefficient memory boundary.

Single-Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.HLD`, and `FIR_DBG_CTL.MEM` bits control the FIR MAC units.

Emulation Considerations

In FIR debug mode, the DMA operations are not observable.

Interrupts

The *FIR Interrupt Overview* table provides the source of interrupt and service instructions for the FIR interrupts.

Accelerator Mode	Default Programmable Interrupt	Sources	Masking	Service
Legacy Mode	FIR_DMA	Window Complete	N/A	ROC from FIR_DMASTAT + RTI instruction
		All channels complete		
	FIR_STAT	MAC IEEE floating point exceptions	N/A	ROC from FIR_MACSTAT + RTI instruction
		MAC fixed point overflow		
Auto Configuration Mode	FIR_DMA	Window Complete	FIR_CTL2 . IMASK	ROC from FIR_DMASTAT + RTI instruction
		FIR_STAT		
	FIR_STAT	MAC IEEE floating point exceptions	N/A	ROC from FIR_MACSTAT + RTI instruction
		MAC fixed point overflow		

Sources

The FIR module drives two interrupt signals: `FIR_DMA` for the DMA status and `FIR_STAT` for the MAC status. The FIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

In legacy mode, if the `FIR_CTL1 . CCINTR` bit is set, an interrupt is generated after completion of window processing of each channel. In ACM, the interrupt generation can be selectively masked using the `FIR_CTL2 . IMASK` bit for each channel.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. Note that the interrupt follows the access completion rule, where the interrupt is generated when all data are written back to system memory.

The All Channels Complete interrupt source is only applicable in legacy mode. If the `FIR_CTL1 . CCINTR` bit is not set in legacy mode, all channel complete interrupt is generated. In ACM, interrupt generation for each channel is controlled using the `FIR_CTL2 . IMASK` bit.

NOTE: The `FIR_CTL1 . CCINTR` bit is valid only in legacy mode. In ACM, interrupt generation for each channel is controlled using the `FIR_CTL2 . IMASK` bit.

MAC Status

A MAC status interrupt is generated under the following conditions and is reflected in the [FIR_MACSTAT](#) register.

- Multiplier result zero – Set if multiplier result is zero
- Multiplier result infinity – Set if multiplier result is infinity
- Multiply invalid – Set if multiply operation is invalid
- Adder result zero – Set if adder result is zero
- Adder result infinity – Set if adder result is infinity
- Adder invalid – Set if addition is invalid
- Adder overflow – for fixed-point operation

Service

The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this state by reading the MAC status register ([FIR_MACSTAT](#)). The read of the register clears the (sticky) MAC interrupt status bits.

The status interrupt sources are derived from the [FIR_MACSTAT](#) register. A status interrupt can occur due to the last set of MAC operations of a processing iteration that correspond to a particular channel. The interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped when another processing iteration results in a non-zero or valid multiply or add result.

ADSP-2156x FIR Register Descriptions

The FIR accelerator (FIR) contains the following registers.

Table 38-8: ADSP-2156x FIR Register List

Name	Description
FIR_CHNPTR	FIR Chain Pointer Register
FIR_COEFCNT	FIR Coefficient Count Register
FIR_COEFIDX	FIR Coefficient Index Register
FIR_COEFMOD	FIR Coefficient Modifier Register
FIR_CTL1	FIR Global Control Register
FIR_CTL2	FIR Channel Control Register
FIR_DBG_ADDR	Debug Address Register
FIR_DBG_CTL	FIR Debug Control Register
FIR_DBG_RDDAT	FIR Debug Data Read Register
FIR_DBG_WRDAT	FIR Debug Data Write Register
FIR_DMASTAT	FIR DMA Status Register

Table 38-8: ADSP-2156x FIR Register List (Continued)

Name	Description
FIR_INBASE	FIR Input Data Base Register
FIR_INCNT	FIR Input Data Count Register
FIR_INIDX	FIR Input Data Index Register
FIR_INMOD	FIR Input Data Modifier Register
FIR_MACSTAT	FIR MAC Status Register
FIR_OUTBASE	FIR Output Data Base Register
FIR_OUTCNT	FIR Output Data Count Register
FIR_OUTIDX	FIR Output Data Index Register
FIR_OUTMOD	FIR Output Data Modifier Register
FIR_SCTL1	software control register 1
FIR_SCTL2	software control register 2
FIR_SGCTL	Secondary global control register

FIR Chain Pointer Register

The `FIR_CHNPTR` register contains the chain pointer address.

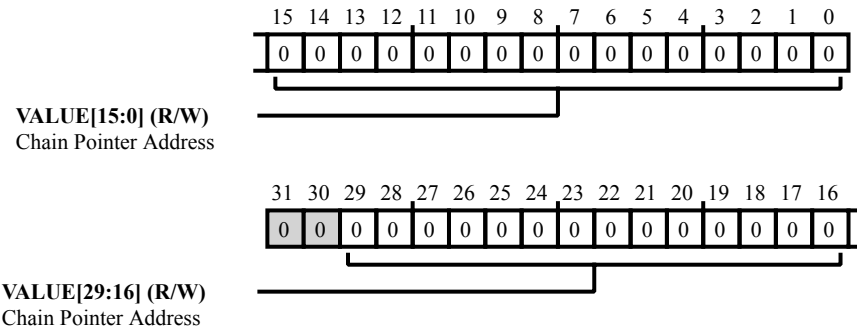


Figure 38-8: FIR_CHNPTR Register Diagram

Table 38-9: FIR_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Chain Pointer Address. The <code>FIR_CHNPTR.VALUE</code> bit field contains the chain pointer address.

FIR Coefficient Count Register

The `FIR_COEFCNT` register contains the 16-bit coefficient buffer count.

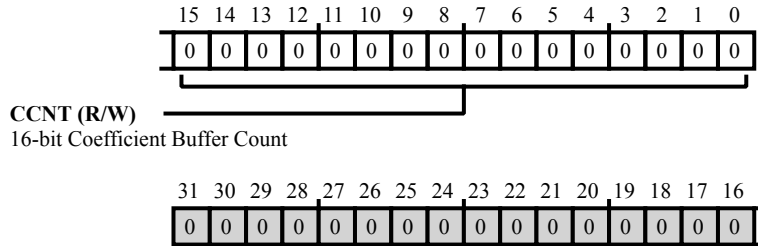


Figure 38-9: FIR_COEFCNT Register Diagram

Table 38-10: FIR_COEFCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CCNT	16-bit Coefficient Buffer Count. The <code>FIR_COEFCNT</code> .CCNT bit field contains the 16-bit coefficient buffer count.

FIR Coefficient Index Register

The `FIR_COEFIDX` register contains the coefficient index word address with the lower two bits removed.

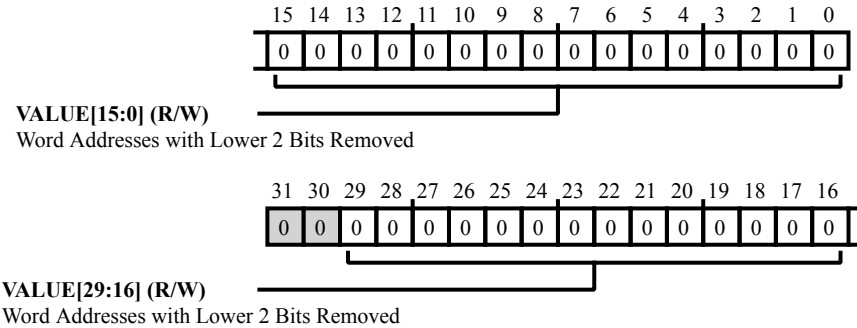


Figure 38-10: FIR_COEFIDX Register Diagram

Table 38-11: FIR_COEFIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Addresses with Lower 2 Bits Removed. The <code>FIR_COEFIDX.VALUE</code> bit field contains the word addresses with the lower 2 bits removed.

FIR Coefficient Modifier Register

The `FIR_COEFMOD` register contains the 16-bit coefficient index modifier.

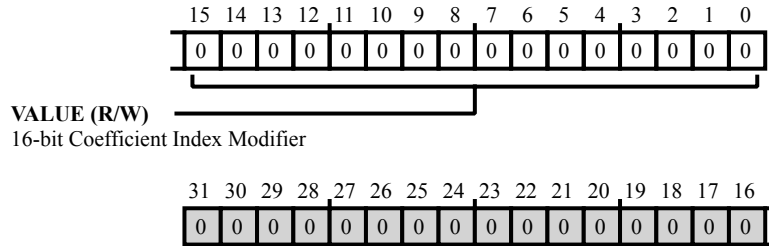


Figure 38-11: FIR_COEFMOD Register Diagram

Table 38-12: FIR_COEFMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Coefficient Index Modifier. The <code>FIR_COEFMOD.VALUE</code> bit field contains the 16-bit coefficient index modifier.

FIR Global Control Register

The `FIR_CTL1` register is used to configure the global parameters for the accelerator. These parameters include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

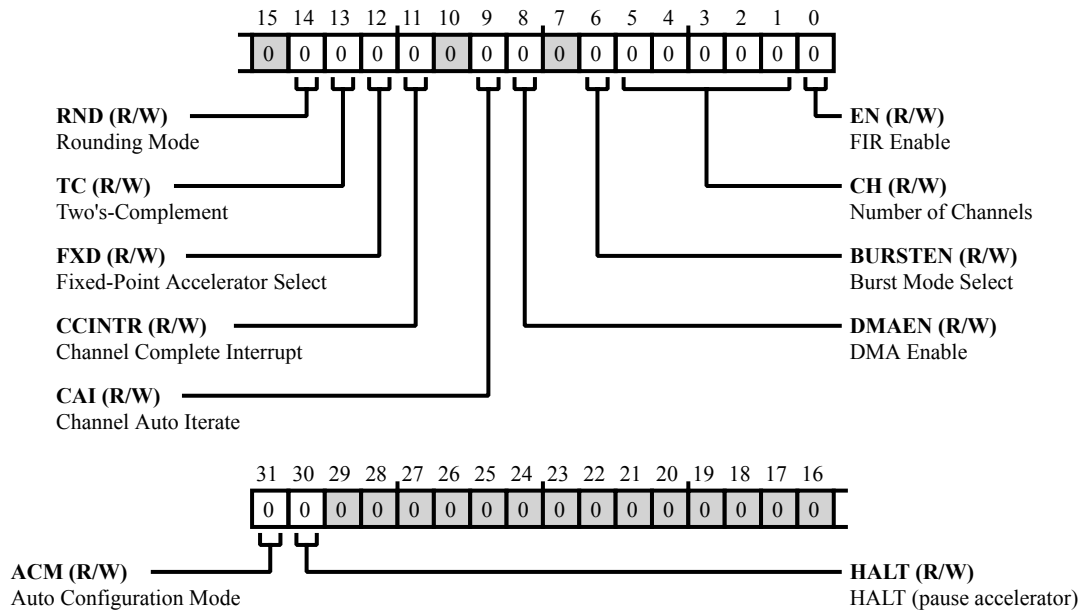


Figure 38-12: FIR_CTL1 Register Diagram

Table 38-13: FIR_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ACM	Auto Configuration Mode. The <code>FIR_CTL1.ACM</code> bit configures the mode for loading the TCB.
		0 Legacy Mode. Load TCB without the <code>FIR_CTL1</code> register. Core intervention may be required.
		1 Auto Configuration Mode. Load TCB with the <code>FIR_CTL1</code> register. Core intervention is not required.
30 (R/W)	HALT	HALT (pause accelerator). This bit is only valid in Auto Configuration mode (ACM)
		0 Release. Release accelerator for further processing of data.
		1 Pause. Pause accelerator for core to check status and modify or submit jobs.

Table 38-13: FIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	RND	Rounding Mode. The FIR_CTL1 . RND bit configures the accelerator to use a rounding mode.
		0 Round to Nearest
		1 Truncate (round towards zero)
13 (R/W)	TC	Two's-Complement. The FIR_CTL1 . TC bit configures the accelerator to use either unsigned integer or signed integer
		0 Unsigned integer
		1 Signed integer
12 (R/W)	FXD	Fixed-Point Accelerator Select. The FIR_CTL1 . FXD bit configures the accelerator to use either 32-bit IEEE floating-point or 32-bit fixed-point.
		0 32-bit IEEE floating-point
		1 32-bit fixed point
11 (R/W)	CCINTR	Channel Complete Interrupt. The FIR_CTL1 . CCINTR bit configures the accelerator to generate an interrupt when each or all channels are done. This bit is only valid in Legacy Mode. In Auto Configuration Mode (ACM), interrupt generation for each channel is controlled using the FIR_CTL2.IMASK bit.
		0 Interrupt is generated only when all channels are done
		1 Interrupt is generated after each channel is done
9 (R/W)	CAI	Channel Auto Iterate. The FIR_CTL1 . CAI bit, if cleared, causes the TDM processing to stop (idle) once all channels are done. If set, processing moves to the first channel and continues TDM processing in a loop when all channels are done. Channel Auto Iterate is not available in Auto Configuration Mode (ACM).
		0 TDM processing stops (idle) once all channels are done
		1 Moves to first channel and continues TDM processing in a loop when all channels are done
8 (R/W)	DMAEN	DMA Enable. The FIR_CTL1 . DMAEN bit enables and disables DMA on the FIR accelerator.
		0 DMA disabled
		1 DMA enabled

Table 38-13: FIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	BURSTEN	Burst Mode Select. When the <code>FIR_CTL1.BURSTEN</code> bit is set, burst mode is enabled for coefficient loads and data loads. When cleared, burst mode is disabled. By default, burst mode is disabled.
5:1 (R/W)	CH	Number of Channels. The <code>FIR_CTL1.CH</code> bit field configures the number of channels and is programmable from 0 to 31. This bit field is only valid in Legacy Mode. There is no channel number limitation in Auto Configuration Mode (ACM) as the accelerator keeps processing the TCBs until the chain pointer becomes null.
		0 Channel 1
		1-30 Channel 2-31
		31 Channel 32
0 (R/W)	EN	FIR Enable. The <code>FIR_CTL1.EN</code> bit enables and disables the FIR accelerator.
		0 Disable FIR
		1 Enable FIR

FIR Channel Control Register

The `FIR_CTL2` register is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio. In Auto Configuration Mode(ACM), this register is also used to configure additional channel specific parameters like interrupts and triggers.

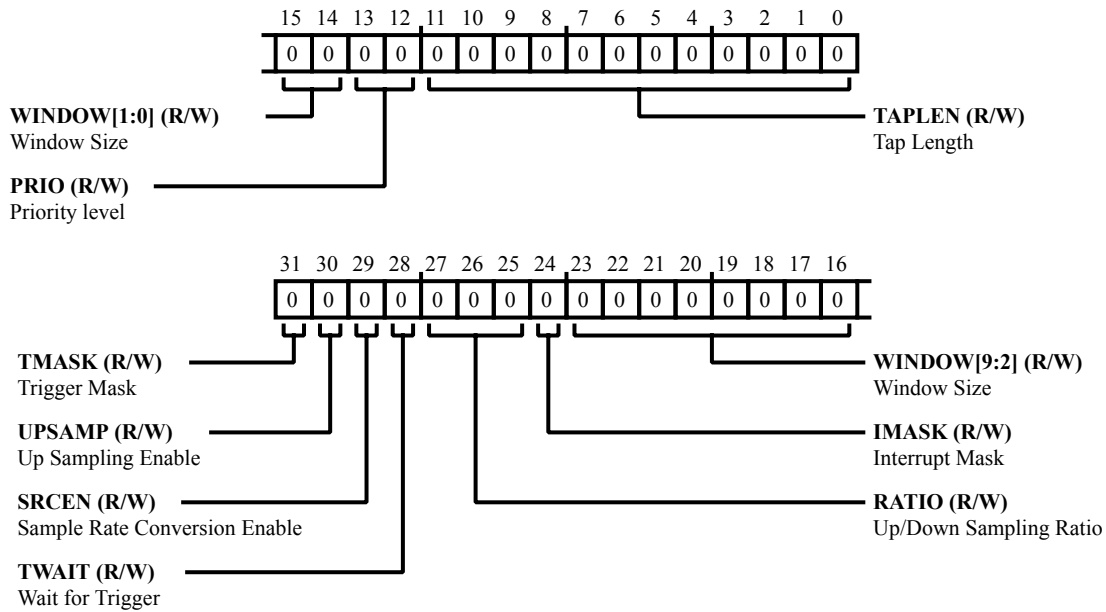


Figure 38-13: FIR_CTL2 Register Diagram

Table 38-14: FIR_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TMASK	Trigger Mask. The <code>FIR_CTL2</code> . <code>TMASK</code> bit enables trigger generation for the channel. It is only valid in Auto Configuration Mode (ACM).
		0 Enable
		1 Mask
30 (R/W)	UPSAMP	Up Sampling Enable. The <code>FIR_CTL2</code> . <code>UPSAMP</code> bit enables up sampling.
		0 Down sampling
		1 Up sampling

Table 38-14: FIR_CTL2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	SRCEN	Sample Rate Conversion Enable. The <code>FIR_CTL2 . SRCEN</code> bit field enables sample rate conversion.
		0 Disabled
		1 Enabled
28 (R/W)	TWAIT	Wait for Trigger. The <code>FIR_CTL2 . TWAIT</code> bit disables the wait for the trigger. It is only valid in Auto Configuration Mode (ACM).
		0 Disable wait for trigger for the channel
		1 Enable wait for the external trigger assertion
27:25 (R/W)	RATIO	Up/Down Sampling Ratio. The <code>FIR_CTL2 . RATIO</code> bit field sets the sampling ratio (<code>FIR_CTL2 . RATIO + 1</code>).
24 (R/W)	IMASK	Interrupt Mask. The <code>FIR_CTL2 . IMASK</code> bit enables interrupt generation for the channel. This bit is only valid in Auto Configuration Mode (ACM).
		0 Enable
		1 Mask
23:14 (R/W)	WINDOW	Window Size. The <code>FIR_CTL2 . WINDOW</code> bit field sets the window size which specifies the number of sample/block to process (sample based processing = window size of 0).
13:12 (R/W)	PRIO	Priority level. The <code>FIR_CTL2 . PRIO</code> bit field indicates the priority.
		0 Level 0 (lowest)
		1 Level 1
		2 Level 2
		3 Level 3 (highest)
11:0 (R/W)	TAPLEN	Tap Length. The <code>FIR_CTL2 . TAPLEN</code> bit field sets the tap length which is programmable between 0-4095 (Tap Length = <code>FIR_CTL2 . TAPLEN + 1</code>).

Debug Address Register

The `FIR_DBG_ADDR` register holds the debug address.

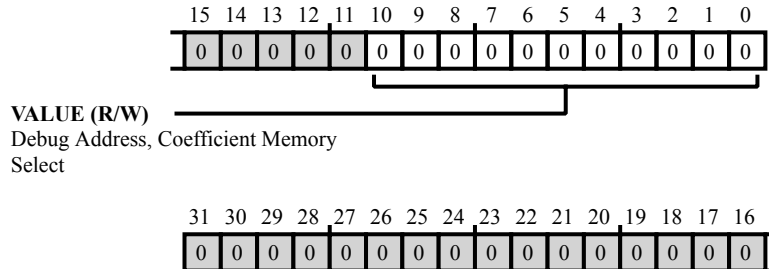


Figure 38-14: `FIR_DBG_ADDR` Register Diagram

Table 38-15: `FIR_DBG_ADDR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Debug Address, Coefficient Memory Select. The <code>FIR_DBG_ADDR.VALUE</code> bit field holds the debug address (bits 0-10). Bit 11 configures whether the memory access is to coefficient memory (=0) or to delay line memory (=1).

FIR Debug Control Register

The `FIR_DBG_CTL` register controls debugging operations such as enabling debug mode running, hold or single stepping.

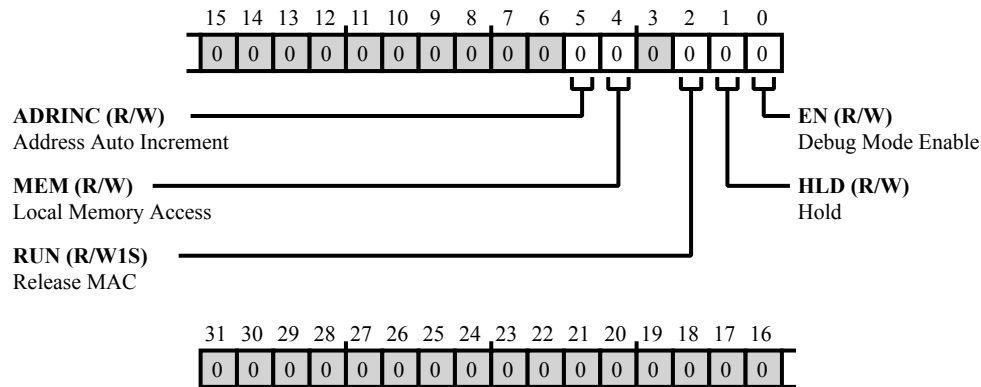


Figure 38-15: FIR_DBG_CTL Register Diagram

Table 38-16: FIR_DBG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	ADRINC	Address Auto Increment. The <code>FIR_DBG_CTL.ADRINC</code> bit allows the address register to auto-increment on <code>FIR_DBG_WRDAT</code> writes and <code>FIR_DBG_RDDAT</code> reads.
4 (R/W)	MEM	Local Memory Access. When the <code>FIR_DBG_CTL.MEM</code> bit is set, the data and coefficients memory can be indirectly accessed.
2 (R/W1S)	RUN	Release MAC. The <code>FIR_DBG_CTL.RUN</code> bit releases the MAC. This bit is self-clearing after one FIR clock cycle.
1 (R/W)	HLD	Hold. The <code>FIR_DBG_CTL.HLD</code> bit holds or single-steps through the FIR.
		0 Hold
		1 Single-step
0 (R/W)	EN	Debug Mode Enable. The <code>FIR_DBG_CTL.EN</code> bit enables debug mode. For local memory access, the <code>FIR_CTL1</code> register can be cleared.
		0 Disable debug mode
		1 Enable debug mode

FIR Debug Data Read Register

The `FIR_DBG_RDDAT` register hold the debug read data.

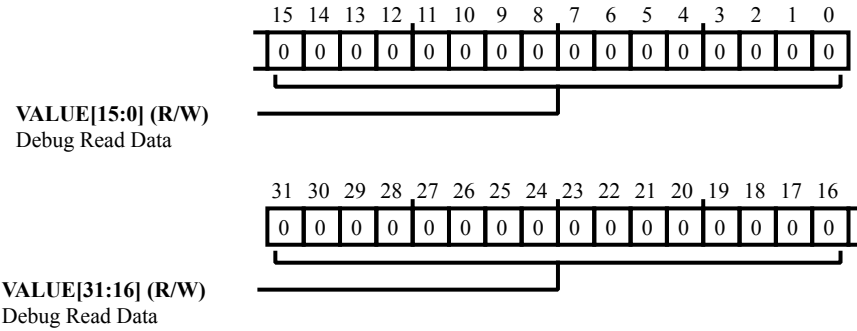


Figure 38-16: FIR_DBG_RDDAT Register Diagram

Table 38-17: FIR_DBG_RDDAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Read Data. The <code>FIR_DBG_RDDAT.VALUE</code> bit field holds the debug read data.

FIR Debug Data Write Register

The `FIR_DBG_WRDAT` register holds the debug write data.

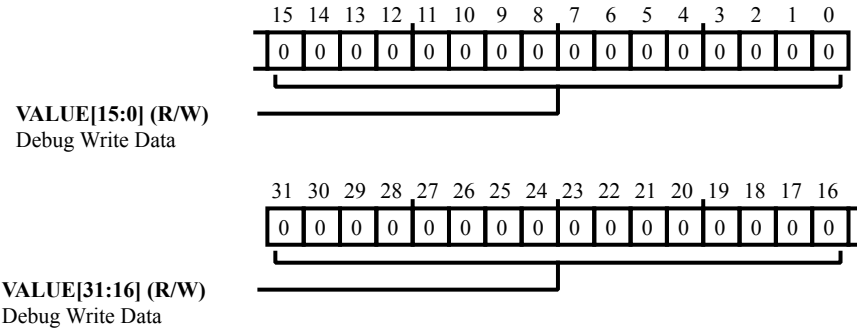


Figure 38-17: `FIR_DBG_WRDAT` Register Diagram

Table 38-18: `FIR_DBG_WRDAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Write Data. The <code>FIR_DBG_WRDAT.VALUE</code> bit field holds the debug write data.

FIR DMA Status Register

The `FIR_DMASTAT` register provides information about chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window complete, all channels complete.

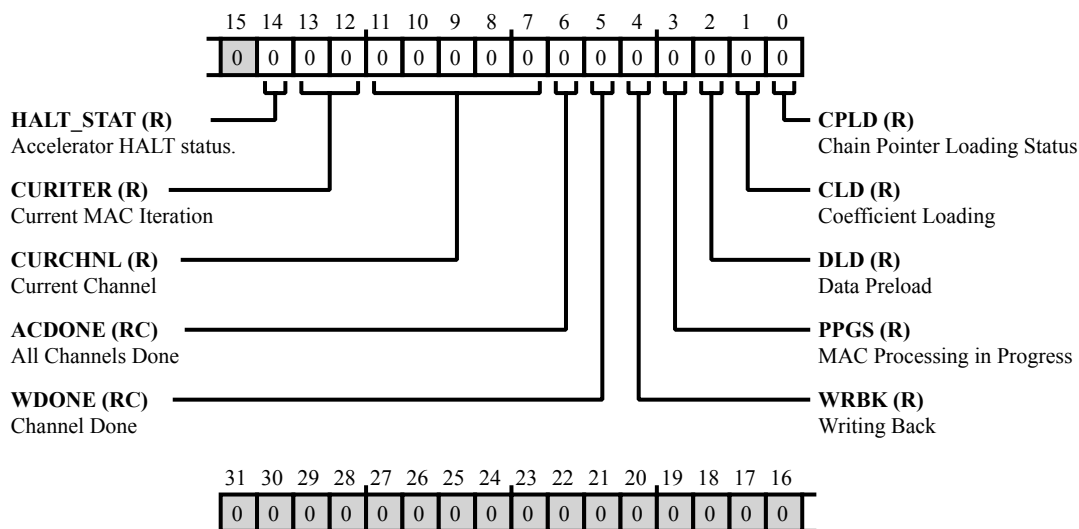


Figure 38-18: FIR_DMASTAT Register Diagram

Table 38-19: FIR_DMASTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	HALT_STAT	Accelerator HALT status. Accelerator HALT status. 1 - Acknowledge to core that acceleration is in halt state. 0 - Not in halt state
13:12 (R/NW)	CURITER	Current MAC Iteration. The <code>FIR_DMASTAT.CURITER</code> bit indicates the current MAC iteration in multi-iteration mode. Zero indicates the final iteration.
11:7 (R/NW)	CURCHNL	Current Channel. The <code>FIR_DMASTAT.CURCHNL</code> bit indicates the channel that is being processed in the TDM slot. Zero indicates the last slot.
6 (RC/NW)	ACDONE	All Channels Done. The <code>FIR_DMASTAT.ACDONE</code> bit indicates the accelerator that processing all channels is complete. This is a sticky bit and is cleared on a register read. The <code>FIR_CTL1.CCINTR</code> bit does not affect the <code>FIR_DMASTAT.ACDONE</code> bit.
5 (RC/NW)	WDONE	Channel Done. The <code>FIR_DMASTAT.WDONE</code> bit indicates the accelerator that processing the current channel is complete. This is a sticky bit and is cleared on a register read. The <code>FIR_CTL1.CCINTR</code> bit does not affect the <code>FIR_DMASTAT.WDONE</code> bit.

Table 38-19: FIR_DMASTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	WRBK	Writing Back. The FIR_DMASTAT.WRBK bit indicates the accelerator is writing back the updated index registers.
3 (R/NW)	PPGS	MAC Processing in Progress. The FIR_DMASTAT.PPGS bit indicates MAC processing in progress.
2 (R/NW)	DLD	Data Preload. The FIR_DMASTAT.DLD bit indicates data preloading.
1 (R/NW)	CLD	Coefficient Loading. The FIR_DMASTAT.CLD bit indicates coefficient loading.
0 (R/NW)	CPLD	Chain Pointer Loading Status. The FIR_DMASTAT.CPLD bit indicates the state machine is in chain pointer load state.

FIR Input Data Base Register

The `FIR_INBASE` register contains the input word base address with the lower two bits removed.

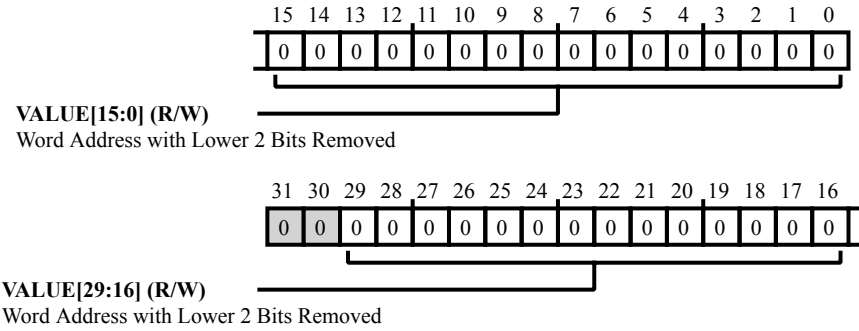


Figure 38-19: FIR_INBASE Register Diagram

Table 38-20: FIR_INBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The <code>FIR_INBASE.VALUE</code> bit field contains the the word address with the lower 2 bits removed.

FIR Input Data Count Register

The `FIR_INCNT` register contains the 16-bit input data count.

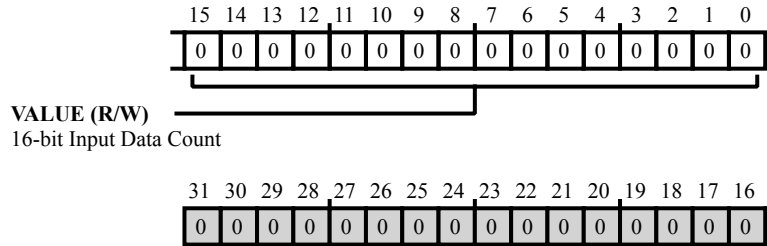


Figure 38-20: FIR_INCNT Register Diagram

Table 38-21: FIR_INCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Count. The <code>FIR_INCNT.VALUE</code> bit field contains the 16-bit input data count.

FIR Input Data Index Register

The `FIR_INIDX` register contains the input word address with the lower two bits removed.

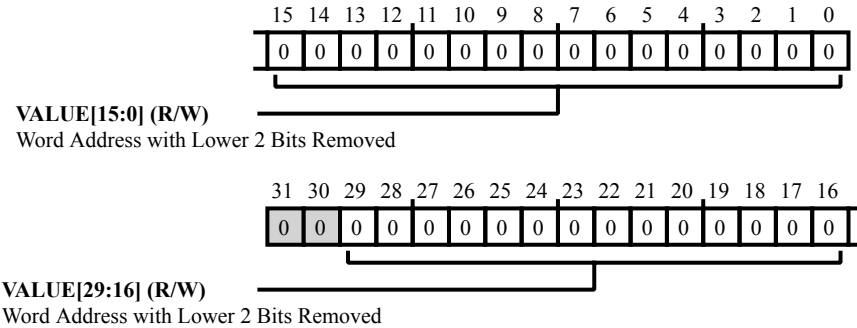


Figure 38-21: FIR_INIDX Register Diagram

Table 38-22: FIR_INIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The <code>FIR_INIDX.VALUE</code> bit field contains the input word address with the lower two bits removed.

FIR Input Data Modifier Register

The `FIR_INMOD` register contains the 16-bit input data buffer modifier.

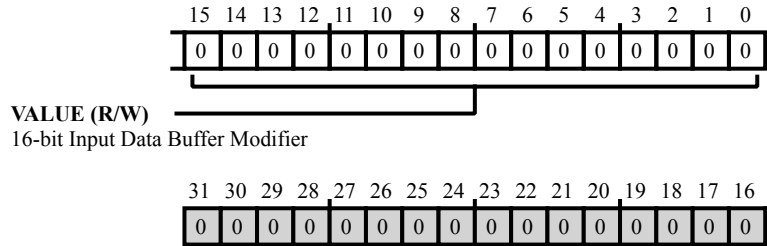


Figure 38-22: FIR_INMOD Register Diagram

Table 38-23: FIR_INMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Modifier. The <code>FIR_INMOD.VALUE</code> bit field contains the 16-bit input data buffer modifier.

FIR MAC Status Register

The `FIR_MACSTAT` register provides the status of MAC operations. The status of all four multipliers/adders are available separately for programs to poll. In fixed-point mode only, the `ARi` bits are used (all other bits are reserved).

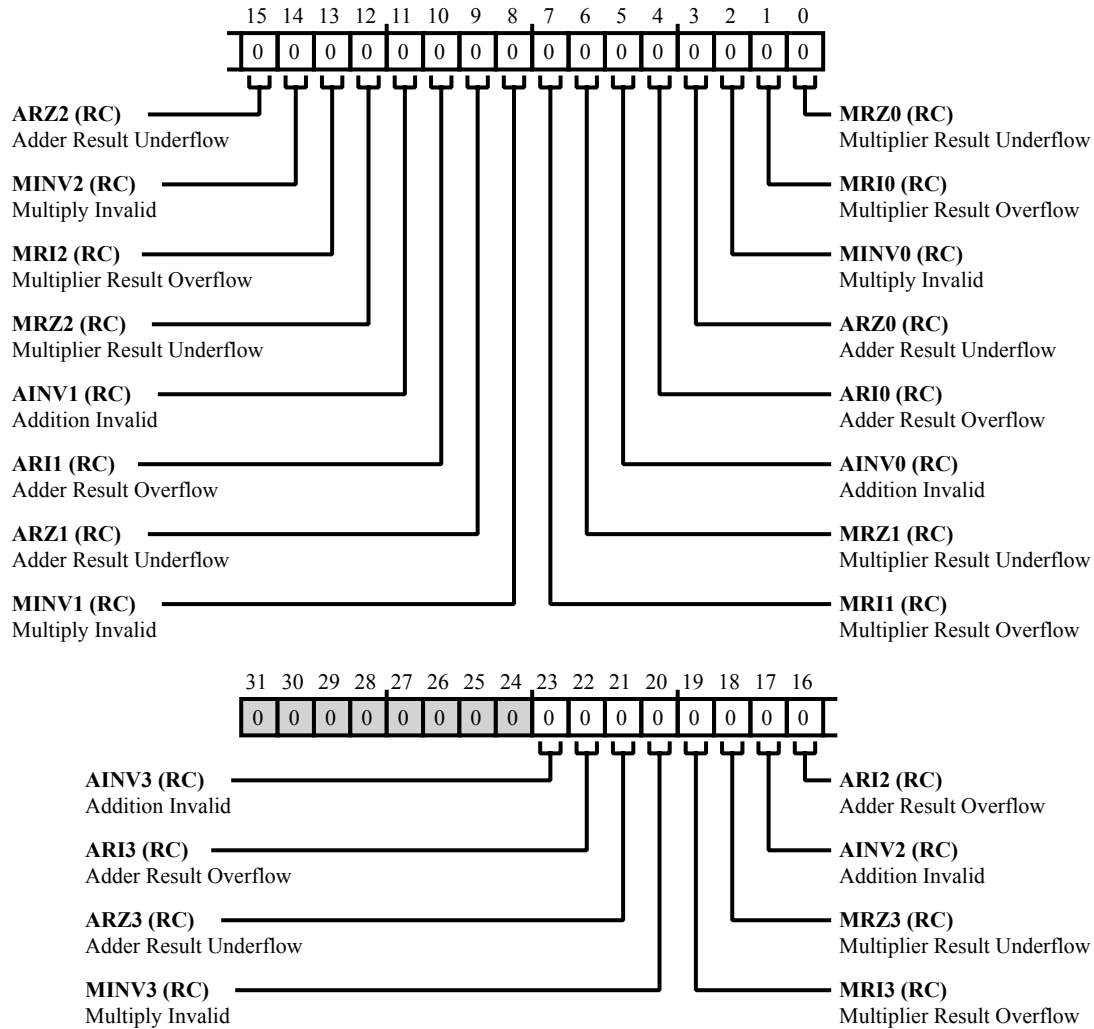


Figure 38-23: FIR_MACSTAT Register Diagram

Table 38-24: FIR_MACSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (RC/NW)	AINV3	Addition Invalid. The <code>FIR_MACSTAT.AINV3</code> bit is set if the adder 3 addition is invalid.

Table 38-24: FIR_MACSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (RC/NW)	ARI3	Adder Result Overflow. The <code>FIR_MACSTAT.ARI3</code> bit is set if the adder 3 result is infinity. Indicates overflow in fixed-point mode.
21 (RC/NW)	ARZ3	Adder Result Underflow. The <code>FIR_MACSTAT.ARZ3</code> bit is set if the adder 3 result is zero.
20 (RC/NW)	MINV3	Multiply Invalid. The <code>FIR_MACSTAT.MINV3</code> bit is set if the multiplier 3 multiply operation is invalid.
19 (RC/NW)	MRI3	Multiplier Result Overflow. The <code>FIR_MACSTAT.MRI3</code> bit is set if the multiplier 3 result is infinity.
18 (RC/NW)	MRZ3	Multiplier Result Underflow. The <code>FIR_MACSTAT.MRZ3</code> bit is set if the multiplier 3 result is zero.
17 (RC/NW)	AINV2	Addition Invalid. The <code>FIR_MACSTAT.AINV2</code> bit is set if the adder 2 addition is invalid.
16 (RC/NW)	ARI2	Adder Result Overflow. The <code>FIR_MACSTAT.ARI2</code> bit is set if the adder 2 result is infinity. Indicates overflow in fixed-point mode.
15 (RC/NW)	ARZ2	Adder Result Underflow. The <code>FIR_MACSTAT.ARZ2</code> bit is set if the adder 2 result is zero.
14 (RC/NW)	MINV2	Multiply Invalid. The <code>FIR_MACSTAT.MINV2</code> bit is set if the multiplier 2 multiply operation is invalid.
13 (RC/NW)	MRI2	Multiplier Result Overflow. The <code>FIR_MACSTAT.MRI2</code> bit is set if the multiplier 2 result is infinity.
12 (RC/NW)	MRZ2	Multiplier Result Underflow. The <code>FIR_MACSTAT.MRZ2</code> bit is set if the multiplier 2 result is zero.
11 (RC/NW)	AINV1	Addition Invalid. The <code>FIR_MACSTAT.AINV1</code> bit is set if the adder 1 addition is invalid.
10 (RC/NW)	ARI1	Adder Result Overflow. The <code>FIR_MACSTAT.ARI1</code> bit is set if the adder 1 result is infinity. Indicates overflow in fixed-point mode.
9 (RC/NW)	ARZ1	Adder Result Underflow. The <code>FIR_MACSTAT.ARZ1</code> bit is set if the adder 1 result is zero.

Table 38-24: FIR_MACSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (RC/NW)	MINV1	<p>Multiply Invalid.</p> <p>The <code>FIR_MACSTAT.MINV1</code> bit is set if the multiplier 1 multiply operation is invalid.</p>
7 (RC/NW)	MRI1	<p>Multiplier Result Overflow.</p> <p>The <code>FIR_MACSTAT.MRI1</code> bit is set if the multiplier 1 result is infinity.</p>
6 (RC/NW)	MRZ1	<p>Multiplier Result Underflow.</p> <p>The <code>FIR_MACSTAT.MRZ1</code> bit is set if the multiplier 1 result is zero.</p>
5 (RC/NW)	AINV0	<p>Addition Invalid.</p> <p>The <code>FIR_MACSTAT.AINV0</code> bit is set if the adder 0 addition is invalid.</p>
4 (RC/NW)	ARI0	<p>Adder Result Overflow.</p> <p>The <code>FIR_MACSTAT.ARI0</code> bit is set if the adder 0 result is infinity. Indicates overflow in fixed-point mode.</p>
3 (RC/NW)	ARZ0	<p>Adder Result Underflow.</p> <p>The <code>FIR_MACSTAT.ARZ0</code> bit is set if the adder 0 result is zero.</p>
2 (RC/NW)	MINV0	<p>Multiply Invalid.</p> <p>The <code>FIR_MACSTAT.MINV0</code> bit is set if the multiplier 0 multiply operation is invalid.</p>
1 (RC/NW)	MRI0	<p>Multiplier Result Overflow.</p> <p>The <code>FIR_MACSTAT.MRI0</code> bit is set if the multiplier 0 result is infinity.</p>
0 (RC/NW)	MRZ0	<p>Multiplier Result Underflow.</p> <p>The <code>FIR_MACSTAT.MRZ0</code> bit is set if multiplier 0 result is zero.</p>

FIR Output Data Base Register

The `FIR_OUTBASE` register contains the output word base address with the lower two bits removed

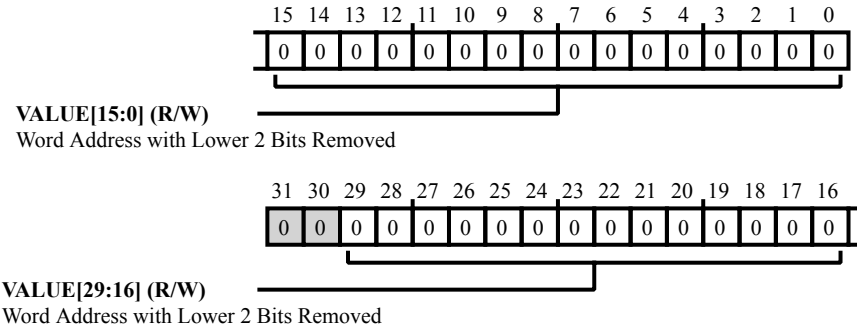


Figure 38-24: FIR_OUTBASE Register Diagram

Table 38-25: FIR_OUTBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The <code>FIR_OUTBASE.VALUE</code> bit field contains the word address with the lower 2 bits removed.

FIR Output Data Count Register

The `FIR_OUTCNT` register contains the 16-bit output buffer count.

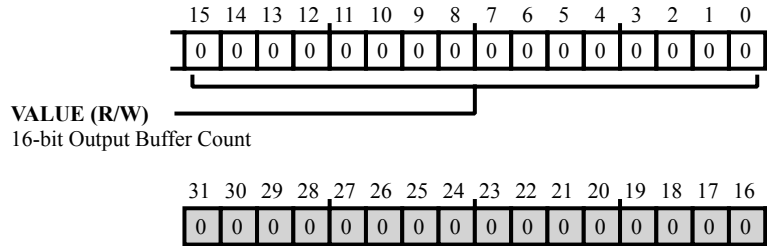


Figure 38-25: FIR_OUTCNT Register Diagram

Table 38-26: FIR_OUTCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Buffer Count. The <code>FIR_OUTCNT.VALUE</code> bit field contains the 16-bit output buffer count.

FIR Output Data Index Register

The `FIR_OUTIDX` register contains the output word address with the lower two bits removed.

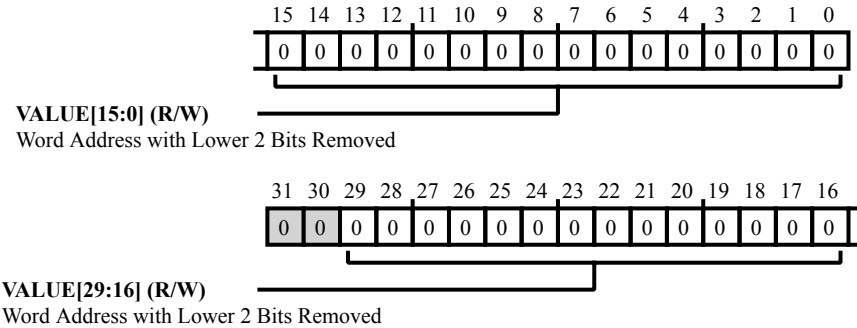


Figure 38-26: FIR_OUTIDX Register Diagram

Table 38-27: FIR_OUTIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The <code>FIR_OUTIDX.VALUE</code> bit field contains the the word address with the lower 2 bits removed.

FIR Output Data Modifier Register

The `FIR_OUTMOD` register contains the 16-bit output data buffer modifier.

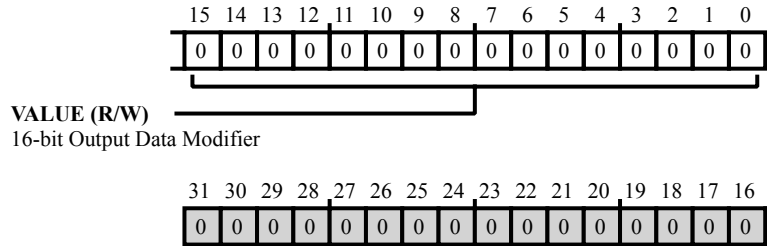


Figure 38-27: FIR_OUTMOD Register Diagram

Table 38-28: FIR_OUTMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Data Modifier. The <code>FIR_OUTMOD.VALUE</code> bit field contains the 16-bit output data modifier.

software control register 1

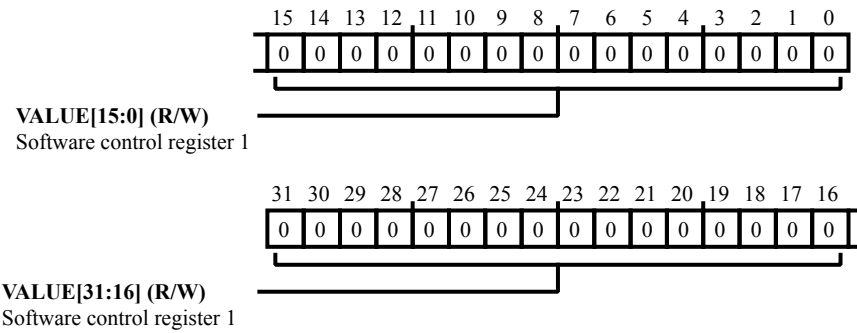


Figure 38-28: FIR_SCTL1 Register Diagram

Table 38-29: FIR_SCTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Software control register 1.

software control register 2

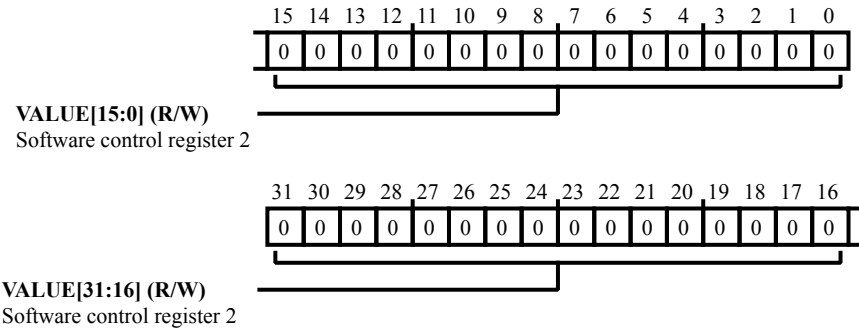


Figure 38-29: FIR_SCTL2 Register Diagram

Table 38-30: FIR_SCTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Software control register 2.

Secondary global control register

The FIRSGCTL register is used to configure the global parameters for the accelerator in ACM mode for loading CTL1 register bits as part of TCB.

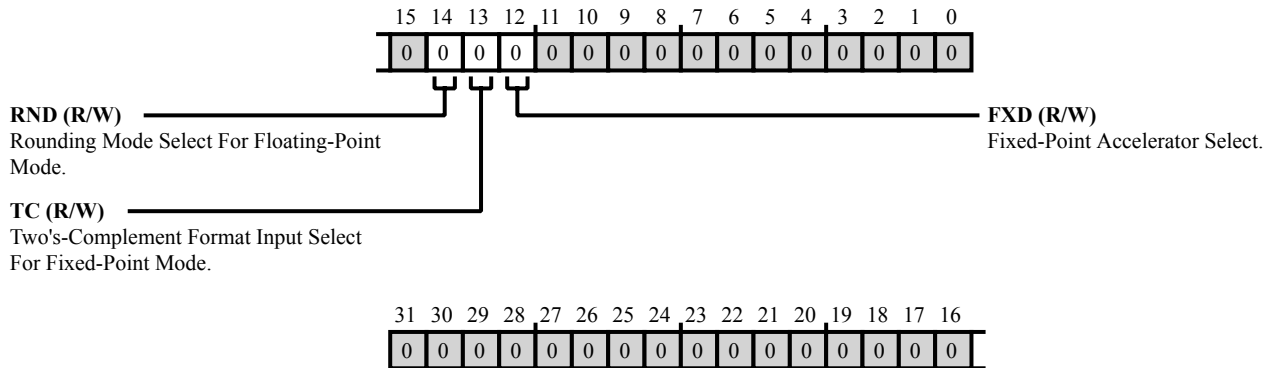


Figure 38-30: FIR_SGCTL Register Diagram

Table 38-31: FIR_SGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	RND	Rounding Mode Select For Floating-Point Mode.. 0- Round to nearest 1 -Truncate (Round towards zero)
13 (R/W)	TC	Two's-Complement Format Input Select For Fixed-Point Mode.. 0 - Unsigned integer 1 - Signed integer
12 (R/W)	FXD	Fixed-Point Accelerator Select.. 0 - 32-bit floating-point compatible to SHARCXI 1 - 32-bit fixed point

39 IIR Accelerator (IIR)

The processor includes an Infinite Impulse Response (IIR) filter accelerator implemented in hardware that reduces the processing load on the core, freeing it up for other tasks.

Features

The accelerator supports a maximum of 32 channels in legacy mode. In Auto Configuration Mode (ACM), there is no limit on number of channels. The accelerator locally stores all the biquad coefficients of 32 channels in legacy mode. Window size can be configured from 1 (sample based) to 1024.

The IIR has the following features:

- Supports IEEE floating point format 32/40-bit
- Rounding modes compatible with SHARC+ core MACs
- Sample-based or window-based processing
- Up to 64 cascaded biquads per channel
- Up to 32 filter channels available in TDM
- Allows biquad save state storage

NOTE: The IIR accelerator module has local memory which is not accessible by the core during regular operation mode. Unlike in the previous SHARC processors, the IIR accelerator modules each have access to the system memory (on-chip/off-chip).

Unlike in the previous SHARC processors, where only one of the IIR or FIR accelerator can be enabled at a time, the processor can use both the IIR and the FIR accelerators at the same time.

Clocking

The IIR accelerator runs at the speed of core clock.

Functional Description

The *IIR Accelerator Block Diagram* shows the various blocks of the IIR hardware accelerator.

The accelerator has:

- a coefficient memory size of 1440×40 bits (288 Biquads x 5 coefficients).
- a data memory size of 576×40 bits (288 Biquads x 2 states).
- one SHARC+ core compatible MAC unit with an input data buffer to supply data to the MAC.

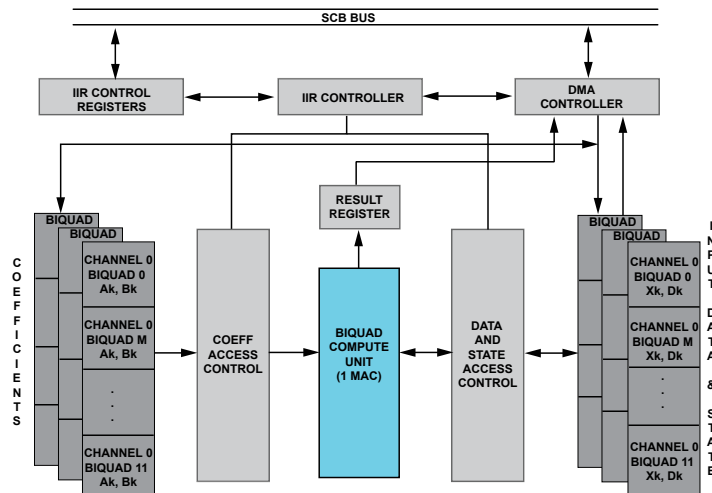


Figure 39-1: IIR Accelerator Block Diagram

The IIR accelerator is implemented using Transposed Direct Form II biquad which has less coefficient sensitivity. The *Transposed Direct Form II Biquad* figure shows the signal flow graph for the biquad structure.

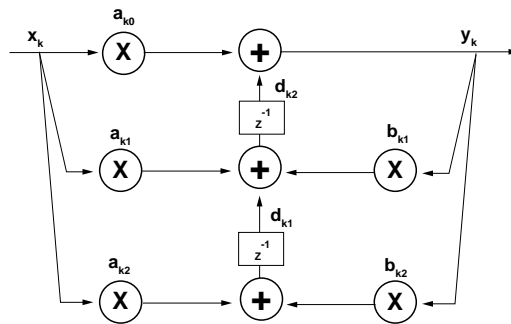


Figure 39-2: Transposed Direct Form II Biquad

The accelerator has the following logical subblocks.

- A datapath unit with the following elements:
 - 32/40-bit coefficient memory (A_k , B_k) for storing biquad coefficients
 - 32/40-bit input data (X_k) and state (D_k)

- One 40/32-bit floating-point multiplier and adder (MAC) unit
- An input data buffer to efficiently supply data to MAC
- One 40-bit result register to hold result of biquad
- Configuration registers for controlling various parameters such as the number of biquads, the number of channels, interrupt control, and DMA control
- A core access interface for writing the DMA/filter configuration registers and for reading the status registers
- A DMA bus interface for transferring data to and from the accelerator. This interface is also used to preload the coefficients (A_k , B_k) and state (D_k) at startup.
- DMA configuration registers for the transfer of input data, output data, and coefficients

The accelerator block is integrated with the SHARC core. For more information about this, see [Figure 41-4 SHARC SCB with FIR/IIR Accelerator](#).

ADSP-2156x IIR Register List

The IIR module reduces the processing load on the core. For more information on IIR functionality, see the IIR register descriptions.

Table 39-1: ADSP-2156x IIR Register List

Name	Description
IIR_CHNPTR	Chain Pointer Register
IIR_COEFIDX	Coefficient Buffer Index Register
IIR_COEFLEN	Coefficient Buffer Length Register
IIR_COEFMOD	Coefficient Index Modifier Register
IIR_CTL1	Global Control Register
IIR_CTL2	Channel Control Register
IIR_DBG_ADDR	IIR Debug Address Register
IIR_DBG_CTL	IIR Debug Control Register
IIR_DBG_RDDAT_HI	IIR Debug Read Data High Register
IIR_DBG_RDDAT_LO	IIR Debug Read Data Low Register
IIR_DBG_WRDAT_HI	IIR Debug Write Data High Register
IIR_DBG_WRDAT_LO	IIR Debug Write Data Low Register
IIR_DMASTAT	DMA Status Register
IIR_INBASE	Input Buffer Base Register
IIR_INIDX	Input Data Index Register
IIR_INLEN	Input Data Buffer Length Register

Table 39-1: ADSP-2156x IIR Register List (Continued)

Name	Description
IIR_INMOD	Input Data Index Modifier Register
IIR_MACSTAT	MAC Status Register
IIR_OUTBASE	Output Buffer Base Register
IIR_OUTIDX	Output Data Buffer Index Register
IIR_OUTLEN	IIR Output Data Buffer Length Register
IIR_OUTMOD	IIR Output Data Index Modifier Register
IIR_SCTL1	Software Control Register1
IIR_SCTL2	Software Control Register2
IIR_SGCTL	Secondary Global Control Register

ADSP-2156x IIR Interrupt List

Table 39-2: ADSP-2156x IIR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
120	IIR0_DMA	IIR0 DMA	Edge	
121	IIR0_STAT	IIR0 Status	Edge	

ADSP-2156x IIR Trigger List

Table 39-3: ADSP-2156x IIR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
40	IIR0_DMA	IIR0 DMA	Edge
104	IIR0_TRGO	IIR0 IIR Trigger output	Edge

Table 39-4: ADSP-2156x IIR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
117	IIR0_TRGI	IIR0 IIR Trigger input	Pulse

Multiply and Accumulate (MAC) Unit

The *IIR MAC Unit* figure shows a pipelined multiplier and accumulator unit that operates on the data and coefficient fetched from the data and coefficient memory. The MAC can perform either 32-bit floating-point or 40-bit floating-point MAC operations. 32-bit floating-point operations generate 32-bit results and 40-bit floating-point operations generate 40-bit results.

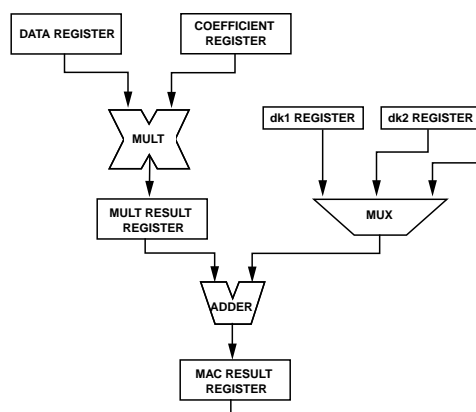


Figure 39-3: IIR MAC Unit

Input Data and Biquad State

The size of data memory is 576×40 bits and is used to hold the $dk1$ and $dk2$ state of all the biquads locally. The DMA controller fetches the sample data from internal memory and calculates the output as well as the $dk1$ and $dk2$ values for each biquad and stores them in local data memory.

Coefficient Memory

The size of coefficient memory is 1440×40 bits and is used to store all the coefficients of all the biquads. At start-up, DMA loads the coefficients from system memory into local coefficient memory.

Internal Memory Storage

This section describes the required storage model for the IIR accelerator.

Coefficient Memory Storage

Coefficients and Dk values for a particular biquad $BQD[k]$ should be stored in internal memory in the order: $Ak0$, $Ak1$, $Bk1$, $Ak2$, $Bk2$, $Dk2$, $Dk1$.

NOTE: The naming convention for the filter coefficients used here is different from the one used in MATLAB. The following conversion should be used when using MATLAB generated coefficients:

$(Akx = bx$ and $Bkx = -ax)$.

Store the coefficients for each biquad in the order:

```
b0, b1, -a1, b2, -a2, dk2, dk1
```

For N biquad stages, store the coefficients in the order:

```
b01, b11, -a11, b21, -a21, dk21, dk11,
b02, b12, -a12, b22, -a22, dk22, dk12,
, . . . . .
b0N, b1N, -a1N, b2N, -a2N, dk2N, dk1N.
```

where bxN and axN are the coefficients ($[b, a]$) for the N th biquad stage.

Operating Modes

The accelerator operates in [Window Processing Mode](#), [40-Bit Floating-Point Mode](#) and [Save Biquad State Mode](#).

Window Processing Mode

Sample-based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window-based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

40-Bit Floating-Point Mode

In 40-bit floating-point mode, the input data/coefficient is treated as a 40-bit floating-point number. 40-bit floating-point MAC operations generate 40-bit results. This mode can be selected by setting the `IIR_CTL1.FORTYBIT` bit.

In ACM, 40-bit floating-point mode can be selected by setting the `IIR_SGCTL.FORTYBIT` field for each channel.

As the DMA bus width is 32 bits, in 40-bit mode the IIR accelerator performs two packed 32-bit accesses to the memory to:

- fetch one 40-bit input or coefficient data, or
- to store one 40-bit output word

The first 32-bit word provides the lower 32 bits and the 8 LSBs of the second 32-bit word provides rest of the upper 8 bits of the complete 40-bit word. The *32-Bit to 40-Bit Packing* figure shows the 32–40 bit packing used by accelerator.

NOTE: Overheads could be required to pack the input 40-bit data into the format acceptable by the IIR accelerator and for unpacking the output of accelerator to the format acceptable by the rest of the application.

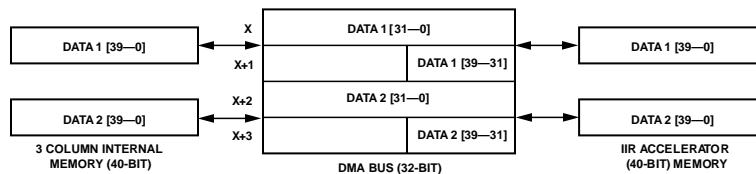


Figure 39-4: 32-Bit to 40-Bit Packing

Save Biquad State Mode

The `IIR_CTL1.SS` bit completely stores the current biquad states in local memory (writes all the DK1 and DK2 states back into the system memory states). This functionality is useful in applications that require fast switching to another high-priority accelerator task—a required IIR to FIR processing transition for example. After resuming,

these states can be reloaded and IIR processing can be continued. DMA status is automatically stored after each iteration.

NOTE: In legacy mode, the accelerator loads the biquad coefficients for all the channels before processing the first channel. After processing the last channel, if `IIR_CTL1 . SS` bit is set, the accelerator stores the biquad states of all the channels in local memory (writes all the DK1 and DK2 states back into the system memory states). In ACM, the accelerator loads the biquad coefficients for only the current channel before starting processing of the current channel. If the `IIR_SGCTL . SS` bit is set, the accelerator stores the biquad states of the current channel in local memory before moving on to the next channel.

NOTE: Write access to any of the IIR accelerator registers loaded by chaining is not allowed while the save state operation is in progress. Attempted writes to these registers could result in the blocking of IOP core reads until the save state operation completes.

Auto Configuration Mode (ACM)

The accelerator can be operated in the legacy mode or Auto Configuration Mode (ACM), controlled by the `IIR_CTL1 . ACM` bit. The default functional mode is legacy mode. The accelerator mode can only be changed when the accelerator is disabled.

The ACM provides the following additional features:

- Halt and Queuing

Core may pause the current TCB chain being processed by setting the `IIR_CTL1 . HALT` bit. The accelerator acknowledges the core by setting the `IIR_DMASTAT . HALT_STAT` bit. The core can appropriately take action to submit/insert new TCB's. After the core takes action, the accelerator processing can be resumed by clearing the `IIR_CTL1 . HALT` bit. Before halting the accelerator, if the initial TCB chain is processed, the accelerator comes to idle state. In this case, the accelerator has to be disabled and enabled by toggling the `IIR_CTL1 . EN` bit and the `IIR_CTL1 . HALT` bit has to be cleared to resume processing.

- No Channel Number Limitation

Unlike in legacy mode, there is no fixed channel number limitation and, therefore, the accelerator ignores the value programmed in the `IIR_CTL1 . CH` field. The application can queue unlimited number of channels/TCBs dynamically and the accelerator keeps processing the TCBs until the chain pointer becomes null.

- Selective Interrupt

Core can enable/mask interrupt generation for each channel using the `IIR_CTL2 . IMASK` bit. If the bit is cleared, an interrupt is generated after completion of the channel.

- Selective Master/Slave Trigger

Core can enable/mask trigger generation by the accelerator after the end of processing of each channel using the `IIR_CTL2 . TMASK` bit. The accelerator can also wait for a trigger after loading the TCB and coefficients and before processing a channel for which the `IIR_CTL2 . TWAIT` bit is set.

In addition to the above features, there are three additional fields as part of the TCB `IIR_SCTL1`, `IIR_SCTL2`, and `IIR_SGCTL`. The `IIR_SGCTL` can be used to change the `IIR_CTL1` parameters such as rounding mode, forty bit mode, and save state for each channel. The `IIR_SCTL1` and `IIR_SCTL2` can be used as general-purpose registers.

Data Transfers

The IIR filter works exclusively through DMA.

IIR Accelerator TCB

The location of the DMA parameters for the next sequence comes from the chain pointer register. This register points to the next set of DMA parameters stored in the system memory of the processor known as TCB. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Assignment

The structure of a TCB is conceptually the same as the structure of a traditional linked list. Each TCB has several data values and a pointer to the next TCB. The chain pointer of a TCB can point to itself to re-run the same DMA continuously. The IIR accelerator reads each word of the TCB and loads it into the corresponding register. A TCB with a chain pointer register value of zero indicates the end of the chain (no further TCBs are loaded). The IIR accelerator supports circular buffer chained DMA. The following tables show the required TCBs for chained DMA in legacy mode and ACM. TCB refers to the start address of the TCB array.

NOTE: In the IIR accelerator DMA, two different TCB loading sequences are available: one TCB loads five parameters for the coefficients (`IIR_CTL2`, `IIR_COEFIDX`, `IIR_COEFMOD`, `IIR_COEFLEN`, and `IIR_CHNPTR`). The second loads 10 parameters for the data (`IIR_CTL2`, `IIR_INIDX`, `IIR_INMOD`, `IIR_INLEN`, `IIR_INBASE`, `IIR_OUTIDX`, `IIR_OUTMOD`, `IIR_OUTLEN`, `IIR_OUTBASE`, and `IIR_CHNPTR`).

Initialize `IIR_CHNPTR` to `TCB+12` in legacy mode and `TCB + 15` in ACM.

Table 39-5: IIR TCBs for Chained DMA in Legacy Mode

Address	Register
TCB	<code>IIR_CHNPTR</code>
TCB + 0x1	<code>IIR_COEFLEN</code>
TCB + 0x2	<code>IIR_COEFMOD</code>
TCB + 0x3	<code>IIR_COEFIDX</code>
TCB + 0x4	<code>IIR_OUTBASE</code>
TCB + 0x5	<code>IIR_OUTLEN</code>

Table 39-5: IIR TCBs for Chained DMA in Legacy Mode (Continued)

Address	Register
TCB + 0x6	IIR_OUTMOD
TCB + 0x7	IIR_OUTIDX
TCB + 0x8	IIR_INBASE
TCB + 0x9	IIR_INLEN
TCB + 0xA	IIR_INMOD
TCB + 0xB	IIR_INIDX
TCB + 0xC	IIR_CTL2

Table 39-6: IIR TCBs for Chained DMA in ACM

Address	Register
TCB	IIR_CHNPTR
TCB + 0x1	IIR_SCTL1
TCB + 0x2	IIR_SCTL2
TCB + 0x3	IIR_SGCTL
TCB + 0x4	IIR_COEFLEN
TCB + 0x5	IIR_COEFMOD
TCB + 0x6	IIR_COEFIDX
TCB + 0x7	IIR_OUTBASE
TCB + 0x8	IIR_OUTLEN
TCB + 0x9	IIR_OUTMOD
TCB + 0xA	IIR_OUTIDX
TCB + 0xB	IIR_INBASE
TCB + 0xC	IIR_INLEN
TCB + 0xD	IIR_INMOD
TCB + 0xE	IIR_INIDX
TCB + 0xF	IIR_CTL2

DMA Access

The IIR accelerator has two DMA channels (accelerator input and output) to connect to the system memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Burst Mode Access

Burst mode enhances the throughput of the DMA channel and reduce the overall load on the system fabric. If `IIR_CTL1.BURSTEN` is set, all the biquads of all the channels are loaded with burst enabled. The IIR module supports only INCR8 burst.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) must be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters (such as number of biquads, window size) for each channel. In ACM, additional parameter such as interrupt mask, trigger mask, and trigger wait are also available.
- Software control register values in ACM for each channel.
- Secondary control register value in ACM to configure rounding mode, forty bit mode, and save state for each channel.
- DMA parameter register values for the input data
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

In legacy mode, The chain pointer (`IIR_CHNPTR`) field of the last channel's TCB should point to the first channel's TCB. This configuration exists so that when the IIR accelerator is enabled, the module:

1. Loads the coefficients (A_k , B_k) and state variables (D_k) for all the channels into the local coefficient memory of the IIR, and
2. Loops back to the first channel again to start fetching the input data for processing.

In ACM, the chain pointer field of the last channel's TCB need not be pointing to the first channel's TCB.

When the accelerator is enabled, the module:

1. Loads the coefficients (A_k, B_k) and state variables (D_k) only for the current channel being processed.
2. Starts fetching input data for processing.
3. After processing the current channel, if the `IIR_SGCTL.SS` bit is set, it saves the biquad states of the current channel in local memory and moves to the next channel.

In ACM, the accelerator keeps processing the TCB's until the chain pointer becomes null.

The accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the `IIR_INIDX` (input index register) and `IIR_OUTIDX` (output index register) values to the TCB in memory. Then, data processing can begin from where it left off during the next time slot of that channel.

For 32-bit mode, the write-back values for the index registers are equal to $IIRII + W$ and $IIROI + W$.

For 40-bit mode, the write-back values are: $IIR_INIDX + 2 \times W$ and $IIR_OUTIDX + 2 \times W$.

Accelerator input and output channels connect to system memory.

NOTE: The `IIR_CTL2` register is part of the IIR TCB. This configuration allows software to program individual IIR channels having different control attributes.

The above index updates are valid only for legacy mode. In ACM, the `IIR_INIDX` and `IIR_OUTIDX` bit fields of the TCB in memory are updated to `0x00000000` and `0xFFFFFFFF` by the accelerator after processing a window of data.

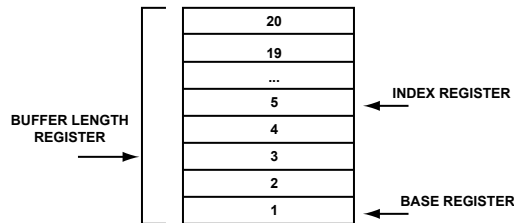


Figure 39-5: Circular Buffer Addressing

Interrupts

The *IIR Interrupt Overview* table provides the source of interrupt and service instructions for the IIR interrupts.

Accelerator Mode	Default Programmable Interrupt	Sources	Masking	Service
Legacy Mode	IIR_DMA	Window Complete	N/A	ROC from <code>IIR_DMASTAT</code> + RTI instruction
		All channels complete		
	IIR_STAT	MAC IEEE floating point exceptions	N/A	ROC from <code>IIR_MACSTAT</code> + RTI instruction
		MAC fixed point overflow		
Auto Configuration Mode	IIR_DMA	Window Complete	<code>IIR_CTL2 . IMASK</code>	ROC from <code>IIR_DMASTAT</code> + RTI instruction
		IIR_STAT		
		MAC fixed point overflow	N/A	

Sources

The IIR module drives two interrupt signals, `IIR_DMA` for the DMA status and `IIR_STAT` for the MAC status. The IIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

In legacy mode, if the `IIR_CTL1.CCINTR` bit is set, an interrupt is generated after completion of window processing of each channel. In ACM, the interrupt generation can be selectively masked using the `IIR_CTL2.IMASK` bit for each channel.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. The interrupt follows the access completion rule, where the interrupt is generated when all data are written back to system memory.

The All Channels Complete interrupt source is only applicable in legacy mode. If the `IIR_CTL1.CCINTR` bit is not set in legacy mode, all channel complete interrupt is generated. In ACM, interrupt generation for each channel is controlled using the `IIR_CTL2.IMASK` bit.

NOTE: In ACM, the Output DMA complete interrupt can be generated after Save State completes for the channel by clearing the `IIR_SGCTL.SSESEL` bit. If this bit is set, the Output DMA interrupt is generated before the Save State completes for the channel.

In Legacy Mode, if the `IIR_CTL1.CCINTR` and `IIR_CTL1.SSESEL` bits are cleared, the All Channel Done Interrupt is generated after the Save State operation is completed for all the channels.

MAC Status

A MAC status interrupt is generated under these conditions:

- Multiplier result zero - Set if multiplier result is zero
- Multiplier result infinity - Set if multiplier result is infinity
- Multiply invalid - Set if multiply operation is invalid
- Adder result zero - Set if adder result is zero
- Adder result infinity - Set if adder result is infinity
- Adder invalid - Set if addition is invalid

Service

The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this state (and clear) by reading the MAC status register (`IIR_MACSTAT`). The MAC interrupt status bits are sticky.

The status interrupt sources are derived from the `IIR_MACSTAT` register. A status interrupt can occur due to the last set of MAC operations of a processing iteration that correspond to a particular channel. The interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped when another processing iteration results in a non-zero or valid multiply or add result.

Programming Model

The following sections provide programming examples in legacy mode and ACM.

Legacy Mode

The IIR accelerator supports up to 32 channels which are time division multiplexed (TDM). Each channel can have a maximum of 64 biquads. The window size for each channel is configurable using the control registers. The maximum configurable window size is 1024.

For multi-channel filtering:

1. Program the number of channels using the `IIR_CTL1.CH` bits.
2. Configure the TCBs in system memory with one channel's TCB pointing to the next channels TCB. The TCB of the last channel should point to the TCB of the first channel.
3. Write the first TCB value into the `IIR_CHNPTR` register and enable the accelerator.
4. If the `IIR_CTL1.BURSTEN` bit is set, ensure that the coefficient buffer end address plus 7 words do not fall outside the valid address range and that the buffer is initialized. When the burst feature is enabled, the IIR module issues an INCR8 burst. When the data count % 8 == 7, the extra 7 words are always read, but ignored by the IIR module.

NOTE: Data count = $\sum(B_n * 7)$ or $\sum(B_n * 14)$; where B_n is number of biquads for channel N. Each biquads contains 7 words for 32-bit mode or 14 words in 40-bit mode.

The *Biquad Processing Program Flow* figure shows the biquad processing program flow.

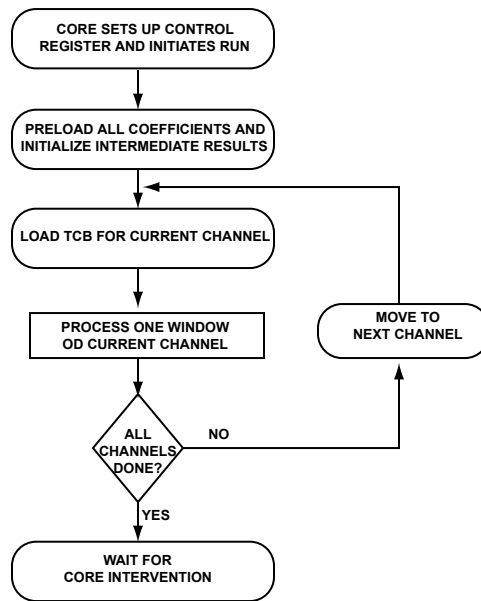


Figure 39-6: Biquad Processing Program Flow

The accelerator uses the following procedure to process biquads:

1. The controller loads all coefficients of all the channels into local storage.
2. Once all the coefficients are loaded, the controller goes to the first biquad of the first channel and calculates the output of the first biquad and updates the intermediate results for that biquad.
3. The accelerator then moves to the next biquad of that channel and repeats the process until all the biquads for that channel are completed and the results are stored to memory.
4. The process is repeated for next sample until one window of the corresponding channel is processed.
5. After one window of the channel is processed, the accelerator moves to the next channel and computes the results.

NOTE: All the addresses programmed in the 32-bit address boundaries and must not contain the lower 2 bits (which are assumed as zeros).

Auto Configuration Mode

In ACM, there is no limit on the number of channels the accelerator can process. The accelerator keeps processing the TCBs until the chain pointer becomes null. Each channel can have a maximum of 64 biquads. Maximum window size configurable is 1024.

The figure shows multi-channel filtering in ACM. Multiple channels are processed in a time multiplexed format (TDM).

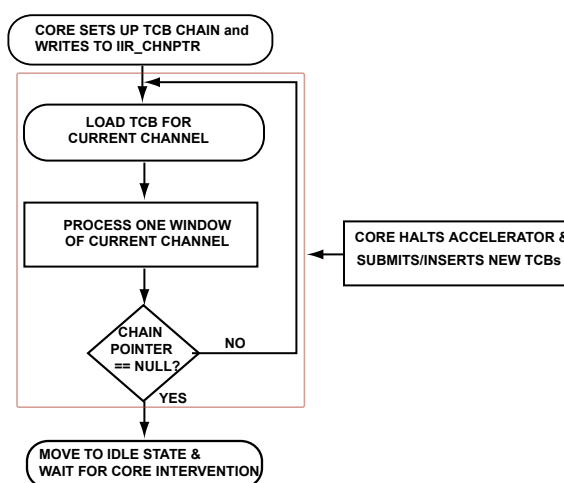


Figure 39-7: Multi-Channel Filtering in Auto Configuration Mode

For multi-channel filtering:

1. Configure the TCBs in system memory with one channel's TCB pointing to the next channel's TCB. There is no limit on the number of channels to be configured. The TCB of the last channel need not be pointing to the TCB of the first channel.
2. Write the first TCB value to the `IIR_CHNPTR` register and enable the accelerator.

The accelerator fetches the first channel's TCB. Using it as pointer, loads the biquad coefficients and biquad states for that channel into its local memory and loads the `IIR_CTL2` and `IIR_SGCTL` registers to configure the filter parameters corresponding to that channel.

If the `IIR_CTL2.TWAIT` bit is set, the accelerator waits for a input trigger to start the window processing for the channel.

The accelerator computes the output of the first biquad and updates the intermediate results for that biquad. Then, the accelerator moves to the next biquad of that channel and repeats the process until all the biquads for that channel are completed and the results are stored to memory. The process is repeated with the next sample until one window of the corresponding channel is processed. If the `IIR_CTL1.SS` bit is set, the accelerator saves the states of the biquads of the channel into system memory. If the `IIR_CTL2.TMASK` bit is cleared the accelerator sends an output trigger after completion of processing the channel.

If the `IIR_CTL2.IMASK` bit is cleared, the accelerator interrupts the core after completion of processing a particular channel.

At the end of the window, the accelerator updates the `IIR_INIDX` and `IIR_OUTIDX` registers to `0x00000000` and `0xFFFFFFFF` in the TCB of system memory and moves to the next channel.

3. At any instant, as required, the core halts the accelerator. It sets the `IIR_CTL1.HALT` bit and appropriately takes action to submit/insert new TCBs and clears the `IIR_CTL1.HALT` bit to resume channel processing.

If the `IIR_CHNPTR` register is zero (last channel is being processed or channel processing is complete) after halting the accelerator, the `IIR_CHNPTR` register is appropriately written before resuming the accelerator channel processing.

If the accelerator is idle after halt, `IIR_CTL1.EN` bit is toggled to disable and re-enable the accelerator and the `IIR_CTL1.HALT` bit is cleared and the accelerator resumes channel processing.

4. The accelerator continues processing until all the channels are complete. Repeat *Step 3* (if required) to submit/insert new channels.

ADSP-2156x IIR Register Descriptions

The IIR filter accelerator (IIR) contains the following registers.

Table 39-7: ADSP-2156x IIR Register List

Name	Description
<code>IIR_CHNPTR</code>	Chain Pointer Register
<code>IIR_COEFIDX</code>	Coefficient Buffer Index Register
<code>IIR_COEFLEN</code>	Coefficient Buffer Length Register
<code>IIR_COEFMOD</code>	Coefficient Index Modifier Register
<code>IIR_CTL1</code>	Global Control Register
<code>IIR_CTL2</code>	Channel Control Register
<code>IIR_DBG_ADDR</code>	IIR Debug Address Register
<code>IIR_DBG_CTL</code>	IIR Debug Control Register
<code>IIR_DBG_RDDAT_HI</code>	IIR Debug Read Data High Register
<code>IIR_DBG_RDDAT_LO</code>	IIR Debug Read Data Low Register
<code>IIR_DBG_WRDAT_HI</code>	IIR Debug Write Data High Register
<code>IIR_DBG_WRDAT_LO</code>	IIR Debug Write Data Low Register
<code>IIR_DMASTAT</code>	DMA Status Register
<code>IIR_INBASE</code>	Input Buffer Base Register
<code>IIR_INIDX</code>	Input Data Index Register
<code>IIR_INLEN</code>	Input Data Buffer Length Register
<code>IIR_INMOD</code>	Input Data Index Modifier Register
<code>IIR_MACSTAT</code>	MAC Status Register
<code>IIR_OUTBASE</code>	Output Buffer Base Register
<code>IIR_OUTIDX</code>	Output Data Buffer Index Register
<code>IIR_OUTLEN</code>	IIR Output Data Buffer Length Register

Table 39-7: ADSP-2156x IIR Register List (Continued)

Name	Description
IIR_OUTMOD	IIR Output Data Index Modifier Register
IIR_SCTL1	Software Control Register1
IIR_SCTL2	Software Control Register2
IIR_SGCTL	Secondary Global Control Register

Chain Pointer Register

The `IIR_CHNPTR` register should be written with word address without the lower 2 bits.

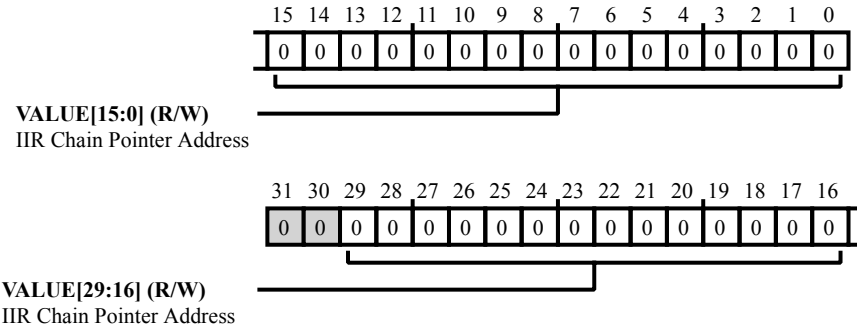


Figure 39-8: IIR_CHNPTR Register Diagram

Table 39-8: IIR_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	IIR Chain Pointer Address. The <code>IIR_CHNPTR.VALUE</code> bit field contains the chain pointer address.

Coefficient Buffer Index Register

The `IIR_COEFIDX` register contains the word address with the lower 2 bits removed.

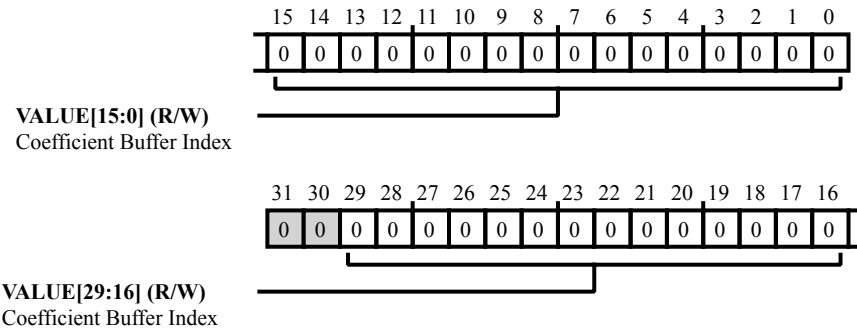


Figure 39-9: IIR_COEFIDX Register Diagram

Table 39-9: IIR_COEFIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Coefficient Buffer Index. The <code>IIR_COEFIDX.VALUE</code> bit field provides the coefficient buffer index.

Coefficient Buffer Length Register

The `IIR_COEFLEN` register provides the coefficient buffer length.

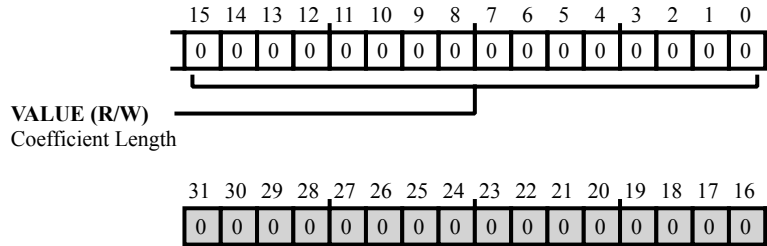


Figure 39-10: IIR_COEFLEN Register Diagram

Table 39-10: IIR_COEFLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Coefficient Length. The <code>IIR_COEFLEN.VALUE</code> bit field provides the coefficient buffer length.

Coefficient Index Modifier Register

The `IIR_COEFMOD` register provides the coefficient index modifier.

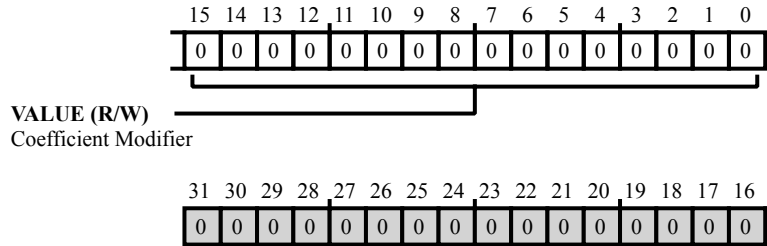


Figure 39-11: IIR_COEFMOD Register Diagram

Table 39-11: IIR_COEFMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Coefficient Modifier. The <code>IIR_COEFMOD.VALUE</code> bit field provides the coefficient modifier.

Global Control Register

The `IIR_CTL1` register is used to configure the global parameters for the accelerator. These parameters include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

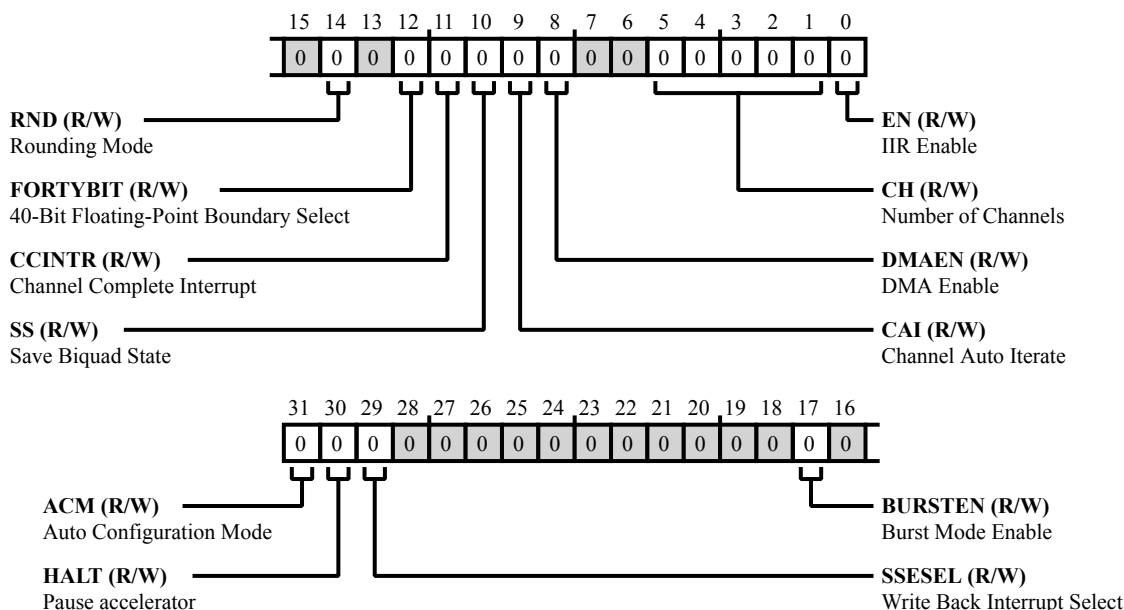


Figure 39-12: `IIR_CTL1` Register Diagram

Table 39-12: `IIR_CTL1` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ACM	Auto Configuration Mode. The <code>IIR_CTL1.ACME</code> bit configures the mode for loading the TCB.
		0 Legacy Mode
		1 Auto Configuration Mode
30 (R/W)	HALT	Pause accelerator. The <code>IIR_CTL1.HALT</code> bit determines whether the accelerator pauses so that the core can check the status and modify or submit a job or the accelerator is released for further processing of data. This bit is only valid in Auto Configuration Mode (ACM).
		0 Release accelerator
		1 Pause accelerator

Table 39-12: IIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	SSESEL	Write Back Interrupt Select. The IIR_CTL1 . SSESEL bit selects whether to generate an interrupt after the write back or after save state completion.
		0 Generate Interrupt after save state completion
		1 Generate interrupt after write back completion
17 (R/W)	BURSTEN	Burst Mode Enable. The IIR_CTL1 . BURSTEN bit field is set, burst access is enabled.
		0 Disable burst mode
		1 Enable burst mode
14 (R/W)	RND	Rounding Mode. The IIR_CTL1 . RND bit field selects the rounding mode for floating-point format.
		0 Round to nearest (even)
		1 Truncate (Round towards zero)
12 (R/W)	FORTYBIT	40-Bit Floating-Point Boundary Select. The IIR_CTL1 . FORTYBIT bit selects between 32-bit IEEE floating-point format or 40-bit IEEE floating-point format.
		0 32-bit IEEE floating-point
		1 40-bit IEEE floating-point
11 (R/W)	CCINTR	Channel Complete Interrupt. The IIR_CTL1 . CCINTR bit configures the channel complete interrupt to generate when all channels are done or after each channel is done. This bit is only valid in legacy mode when the IIR_CTL1.L_TIMASK_EN bit is cleared (=0). In Auto Configuration Mode (ACM) or when the IIR_CTL1.L_TIMASK_EN is set (=1), interrupt generation for each channel is controlled using IIR_CTL2.IMASK bit.
		0 Interrupt is generated only when all channels are done (default)
		1 Interrupt is generated after each channels is done (default)
10 (R/W)	SS	Save Biquad State. The IIR_CTL1 . SS bit configures the accelerator to store the Dk register settings into the internal memory. This can be used to save the biquad states before switching to another high priority accelerator task. This bit is only valid in legacy mode. In Auto Configuration Mode(ACM), Save State for each channel is controlled using IIR_SGCTL.SS bit

Table 39-12: IIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	CAI	Channel Auto Iterate. The <code>IIR_CTL1.CAI</code> bit sets whether TDM processing stops (idle) once all channels complete processing or moves to first channel and continues TDM processing in a loop when all channels complete processing. Channel Auto Iterate is not available in Auto Configuration Mode(ACM). The accelerator keeps processing the TCBs until the chain pointer becomes null in Auto Configuration Mode(ACM).
		0 TDM processing stops (idle) once all channels complete processing
		1 Moves to first channel and continues TDM processing in a loop when all channels complete processing
8 (R/W)	DMAEN	DMA Enable. The <code>IIR_CTL1.DMAEN</code> bit enables DMA on the accelerator.
		0 Disable
		1 Enable
5:1 (R/W)	CH	Number of Channels. The <code>IIR_CTL1.CH</code> bit field configures the number of channels and is programmable between 0-31 (channels = $NCH + 1$). This bit field is only valid in Legacy Mode. In Auto Configuration Mode(ACM), there is no limit on the number of channels. the accelerator keeps processing the TCBs until the chain pointer becomes null.
0 (R/W)	EN	IIR Enable. The <code>IIR_CTL1.EN</code> bit enables or disables the IIR accelerator.
		0 IIR disabled
		1 IIR enabled

Channel Control Register

The `IIR_CTL2` register is used to configure the channel specific parameters. These parameters include the number of biquads and window size. In Auto Configuration Mode(ACM), this register is also used to configure additional channel specific parameters like interrupts and triggers.

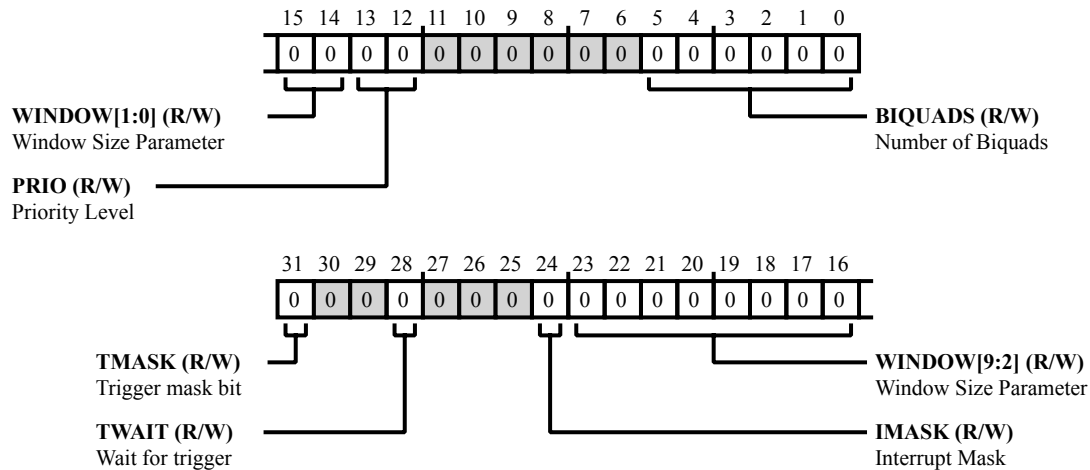


Figure 39-13: IIR_CTL2 Register Diagram

Table 39-13: IIR_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TMASK	Trigger mask bit. The <code>IIR_CTL2</code> . <code>TMASK</code> bit enables trigger generation for the channel. It is valid in Auto Configuration Mode (ACM) or if <code>CTL1.L_TIMASK_EN</code> is set.
		0 Enable
		1 Mask
28 (R/W)	TWAIT	Wait for trigger. The <code>IIR_CTL2</code> . <code>TWAIT</code> bit disables the wait for the trigger. It is valid in Auto Configuration Mode (ACM) or if <code>CTL1.L_TIMASK_EN</code> is set.
		0 Disable wait for the trigger for the channel
		1 Enable wait for external trigger input assertion for the channel
24 (R/W)	IMASK	Interrupt Mask. The <code>IIR_CTL2</code> . <code>IMASK</code> bit enables interrupt generation for the channel. This bit is valid in Auto Configuration Mode (ACM) or if <code>CTL1.L_TIMASK_EN</code> is set.
		0 Enable
		1 Mask

Table 39-13: IIR_CTL2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23:14 (R/W)	WINDOW	Window Size Parameter. The IIR_CTL2 . WINDOW bit field configures the window size which specifies the number of sample/block to process (sample based processing = window size of 1). This bit field should be programmed to "actual window size required -1". For example, for sample based processing this bit field should be programmed to 0.
13:12 (R/W)	PRIO	Priority Level. The IIR_CTL2 . PRIO bit field indicates the priority.
		0 Level 0 (lowest)
		1 Level 1
		2 Level 2
		3 Level 3 (highest)
5:0 (R/W)	BIQUADS	Number of Biquads. The IIR_CTL2 . BIQUADS bit field configures the number of biquads and is programmable between 0-63 (number of Biquads = BIQUADS + 1).

IIR Debug Address Register

The `IIR_DBG_ADDR` register holds the debug address. If bit 11 is set, coefficient memory is selected.

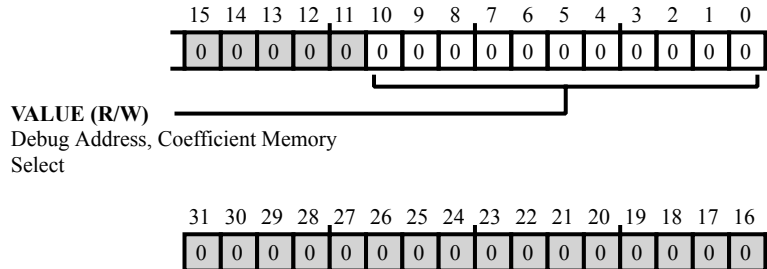


Figure 39-14: IIR_DBG_ADDR Register Diagram

Table 39-14: IIR_DBG_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Debug Address, Coefficient Memory Select. The <code>IIR_DBG_ADDR.VALUE</code> bit field holds the debug address (bits 0-10). Bit 11 configures whether the memory access is to coefficient memory (=0) or to delay line memory (=1).

IIR Debug Control Register

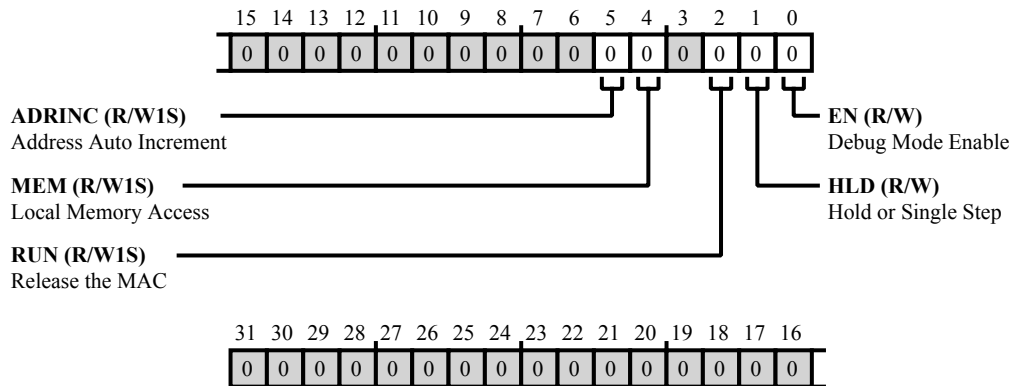


Figure 39-15: IIR_DBG_CTL Register Diagram

Table 39-15: IIR_DBG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1S)	ADRINC	Address Auto Increment. The <code>IIR_DBG_CTL.ADRINC</code> bit allows the address register to auto increment on <code>IIR_DBG_WRDAT_HI/IIR_DBG_WRDAT_LO</code> writes and <code>IIR_DBG_RDDAT_HI/IIR_DBG_RDDAT_LO</code> reads.
4 (R/W1S)	MEM	Local Memory Access. The <code>IIR_DBG_CTL.MEM</code> bit allows the data and coefficients memory to be indirectly accessed.
2 (R/W1S)	RUN	Release the MAC. The <code>IIR_DBG_CTL.RUN</code> bit releases the MAC and is self clearing after one IIR clock cycle.
1 (R/W)	HLD	Hold or Single Step. The <code>IIR_DBG_CTL.HLD</code> bit function is based on the <code>IIR_DBG_CTL.MEM</code> bit setting. For <code>IIR_DBG_CTL.MEM = 0</code> this bit sets single step. For <code>IIR_DBG_CTL.MEM = 1</code> this bit sets hold data.
		0 No effect
0 (R/W)	EN	Debug Mode Enable. The <code>IIR_DBG_CTL.EN</code> bit enables debug mode. For local memory access, the <code>IIR_CTL1</code> register can be cleared.
		0 Disable
		1 Enable

IIR Debug Read Data High Register

The `IIR_DBG_RDDAT_HI` register is part of the 40-bit wide debug mode read data register and holds the upper 8 bits.

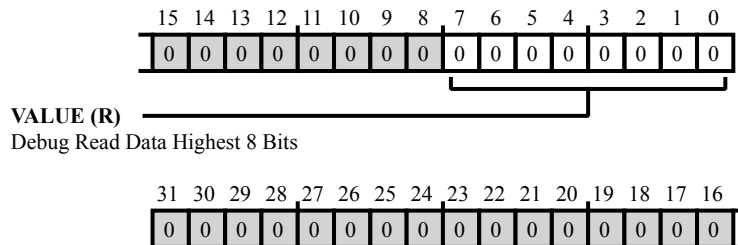


Figure 39-16: IIR_DBG_RDDAT_HI Register Diagram

Table 39-16: IIR_DBG_RDDAT_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Debug Read Data Highest 8 Bits. The <code>IIR_DBG_RDDAT_HI.VALUE</code> bit field holds the upper 8-bit read data.

IIR Debug Read Data Low Register

The `IIR_DBG_RDDAT_LO` register is part of the 40-bit wide debug mode read data register and holds the lower 32 bits.

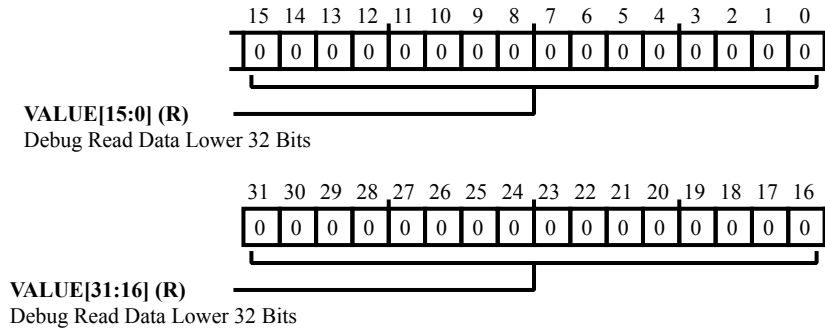


Figure 39-17: IIR_DBG_RDDAT_LO Register Diagram

Table 39-17: IIR_DBG_RDDAT_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Debug Read Data Lower 32 Bits. The <code>IIR_DBG_RDDAT_LO.VALUE</code> bit field holds the lower 32-bit read data.

IIR Debug Write Data High Register

The `IIR_DBG_WRDAT_HI` register is part of the 40-bit wide debug mode write data register and holds the upper 8 bits.

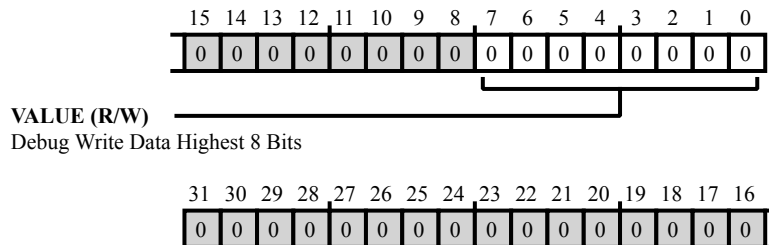


Figure 39-18: IIR_DBG_WRDAT_HI Register Diagram

Table 39-18: IIR_DBG_WRDAT_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Debug Write Data Highest 8 Bits. The <code>IIR_DBG_WRDAT_HI.VALUE</code> bit field holds the upper 8-bit write data.

IIR Debug Write Data Low Register

The `IIR_DBG_WRDAT_LO` register is part of the 40-bit wide debug mode write data register and holds the lower 32 bits.

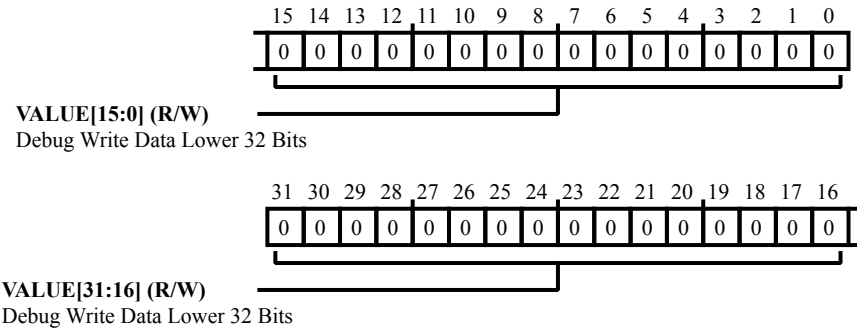


Figure 39-19: IIR_DBG_WRDAT_LO Register Diagram

Table 39-19: IIR_DBG_WRDAT_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Write Data Lower 32 Bits. The <code>IIR_DBG_WRDAT_LO.VALUE</code> bit field holds the lower 32-bit write data.

DMA Status Register

The `IIR_DMASTAT` registers indicate the status of DMA operations.

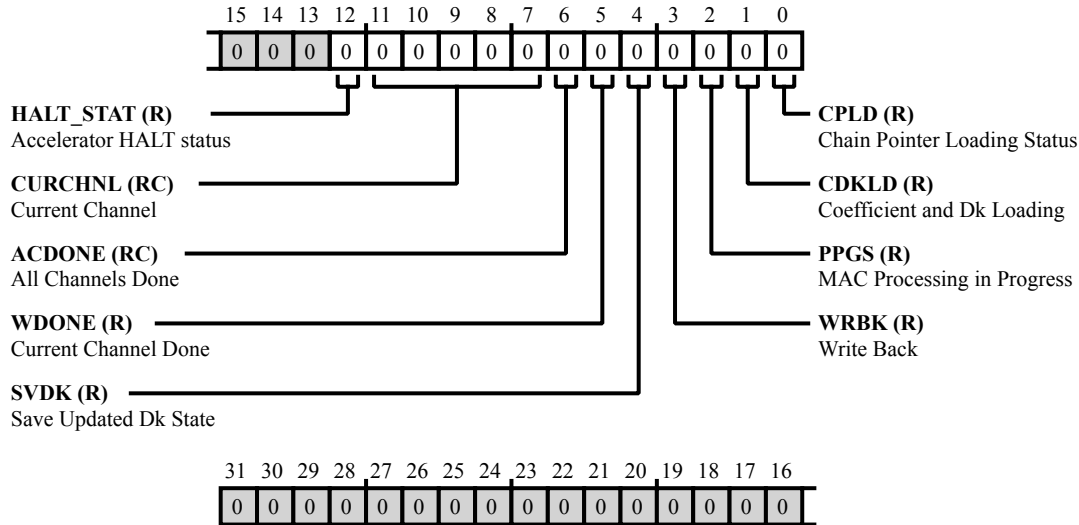


Figure 39-20: IIR_DMASTAT Register Diagram

Table 39-20: IIR_DMASTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/NW)	HALT_STAT	Accelerator HALT status. Acknowledge to core that accelerator is in halt state, if set.
11:7 (RC/NW)	CURCHNL	Current Channel. The <code>IIR_DMASTAT.CURCHNL</code> bit field indicates the channel that is being processed in the TDM slot. Zero indicates the last slot.
6 (RC/NW)	ACDONE	All Channels Done. The <code>IIR_DMASTAT.ACDONE</code> bit indicates all channels are done processing. Note that the <code>IIR_CTL1.CCINTR</code> bit does not affect this status bit. This bit is sticky and is cleared on register read.
5 (R/NW)	WDONE	Current Channel Done. The <code>IIR_DMASTAT.WDONE</code> bit indicates the processing of the current channel is complete. Note that the <code>IIR_CTL1.CCINTR</code> bit does not affect this status bit. This bit is sticky and is cleared on register read.
4 (R/NW)	SVDK	Save Updated Dk State. If there is more than one channel (<code>IIR_CTL1.CH>0</code>), the <code>IIR_DMASTAT.SVDK</code> bit toggles between 0 and 1 as it starts and completes the save state operation on one channel at a time.

Table 39-20: IIR_DMASTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	WRBK	Write Back. The IIR_DMASTAT.WRBK bit indicates the accelerator is writing back updated index registers.
2 (R/NW)	PPGS	MAC Processing in Progress. The IIR_DMASTAT.PPGS bit indicates MAC processing is in progress.
1 (R/NW)	CDKLD	Coefficient and Dk Loading. The IIR_DMASTAT.CDKLD bit indicates the coefficient and Dk are loading.
0 (R/NW)	CPLD	Chain Pointer Loading Status. The IIR_DMASTAT.CPLD bit indicates the IIR is in the chain pointer load state.
		0 State machine not in chain pointer load state
		1 State machine in chain pointer load state

Input Buffer Base Register

The `IIR_INBASE` register contains the word address with the lower 2 bits removed.

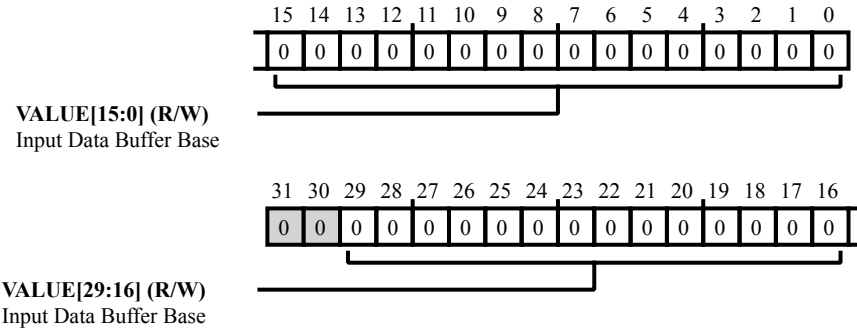


Figure 39-21: IIR_INBASE Register Diagram

Table 39-21: IIR_INBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Input Data Buffer Base. The <code>IIR_INBASE.VALUE</code> bit field value is the input data buffer base address.

Input Data Index Register

The `IIR_INIDX` register contains a word address with the lower 2 bits removed.

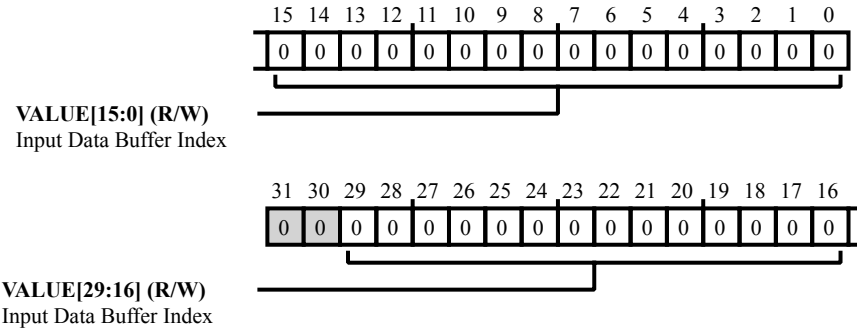


Figure 39-22: IIR_INIDX Register Diagram

Table 39-22: IIR_INIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Input Data Buffer Index. The <code>IIR_INIDX.VALUE</code> bit field value is the input data buffer index.

Input Data Buffer Length Register

The `IIR_INLEN` register provides the input data buffer length.

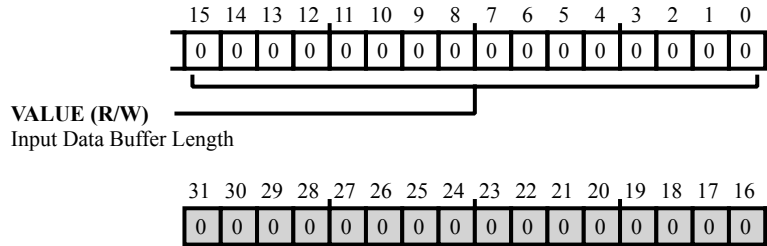


Figure 39-23: IIR_INLEN Register Diagram

Table 39-23: IIR_INLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Input Data Buffer Length. The <code>IIR_INLEN.VALUE</code> bit field value is the input data buffer length.

Input Data Index Modifier Register

The `IIR_INMOD` register provides the 16-bit input data buffer index modifier.

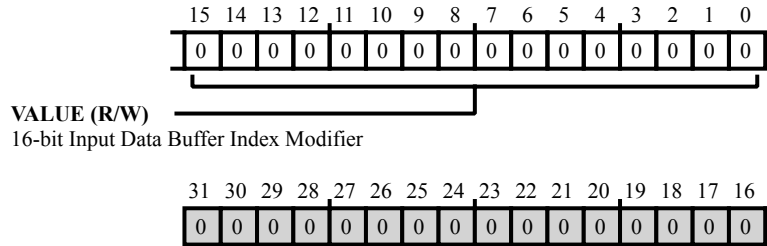


Figure 39-24: IIR_INMOD Register Diagram

Table 39-24: IIR_INMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Index Modifier. The <code>IIR_INMOD.VALUE</code> bit field value is the 16-bit input data buffer modifier.

MAC Status Register

The `IIR_MACSTAT` register indicates the status of MAC operations.

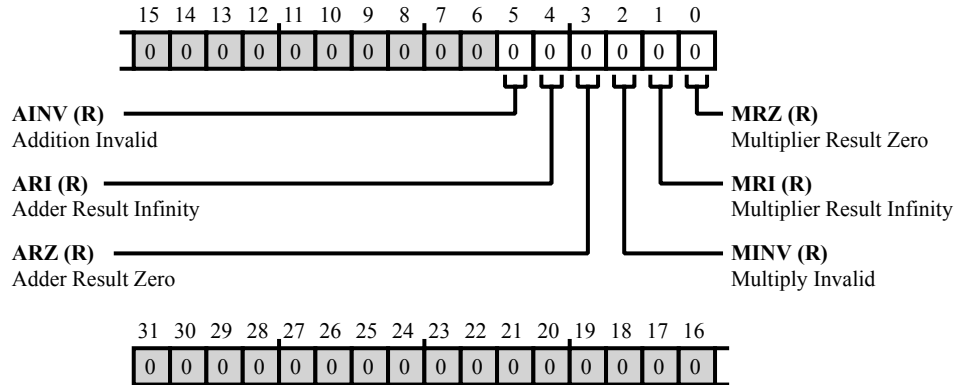


Figure 39-25: IIR_MACSTAT Register Diagram

Table 39-25: IIR_MACSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	AINV	Addition Invalid. The <code>IIR_MACSTAT.AINV</code> bit indicates the addition is invalid.
4 (R/NW)	ARI	Adder Result Infinity. The <code>IIR_MACSTAT.ARI</code> bit indicates the adder result is infinity.
3 (R/NW)	ARZ	Adder Result Zero. The <code>IIR_MACSTAT.ARZ</code> bit indicates the adder result is zero.
2 (R/NW)	MINV	Multiply Invalid. The <code>IIR_MACSTAT.MINV</code> bit indicates the multiply operation is invalid.
1 (R/NW)	MRI	Multiplier Result Infinity. The <code>IIR_MACSTAT.MRI</code> bit indicates the multiplier result is infinity.
0 (R/NW)	MRZ	Multiplier Result Zero. The <code>IIR_MACSTAT.MRZ</code> bit indicates the multiplier result is zero.

Output Buffer Base Register

The `IIR_OUTBASE` register contains the word address with the lower 2 bits removed.

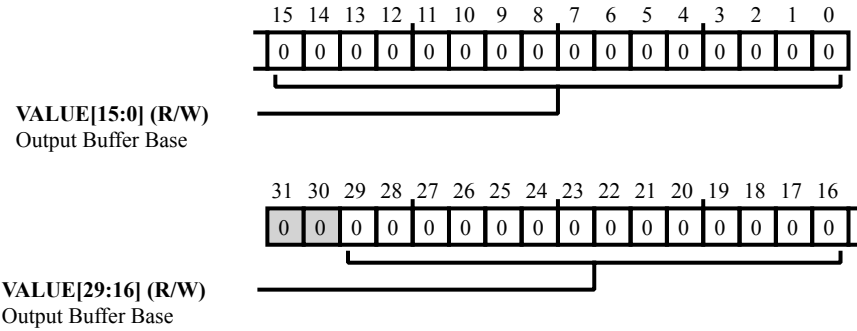


Figure 39-26: IIR_OUTBASE Register Diagram

Table 39-26: IIR_OUTBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Output Buffer Base. The <code>IIR_OUTBASE.VALUE</code> bit field provides the output buffer base address.

Output Data Buffer Index Register

The `IIR_OUTIDX` register should be written with word address without the lower 2 bits

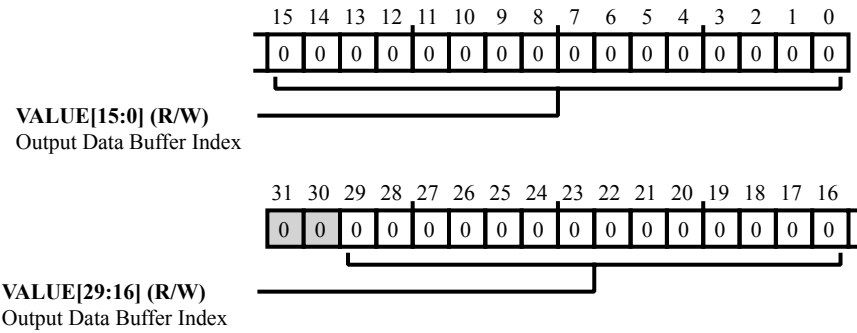


Figure 39-27: IIR_OUTIDX Register Diagram

Table 39-27: IIR_OUTIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Output Data Buffer Index. The <code>IIR_OUTIDX.VALUE</code> bit field provides the output data buffer index.

IIR Output Data Buffer Length Register

The `IIR_OUTLEN` register provides the output data buffer length.

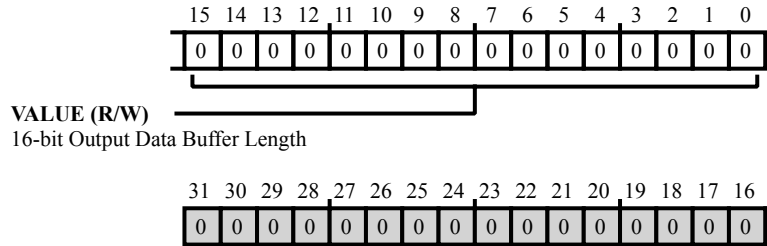


Figure 39-28: IIR_OUTLEN Register Diagram

Table 39-28: IIR_OUTLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Data Buffer Length. The <code>IIR_OUTLEN.VALUE</code> bit field provides the output data buffer length.

IIR Output Data Index Modifier Register

The `IIR_OUTMOD` register provides the output data index modifier.

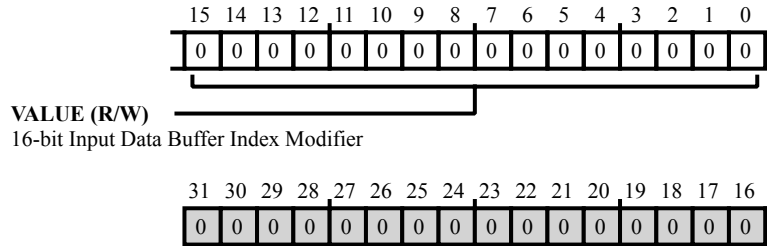


Figure 39-29: IIR_OUTMOD Register Diagram

Table 39-29: IIR_OUTMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Index Modifier. The <code>IIR_OUTMOD.VALUE</code> bit field provides the output data buffer index modifier.

Software Control Register1

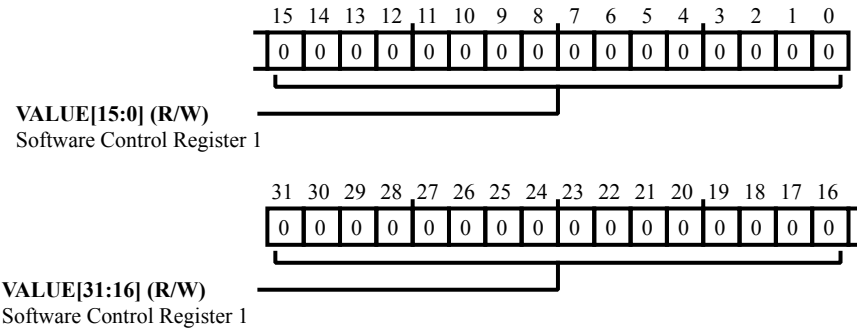


Figure 39-30: IIR_SCTL1 Register Diagram

Table 39-30: IIR_SCTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Software Control Register 1.

Software Control Register2

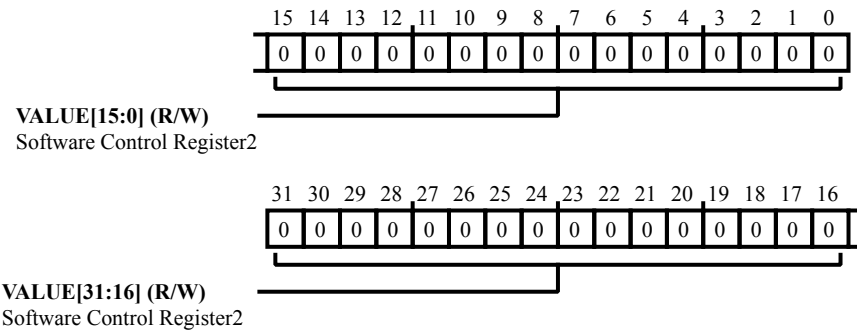


Figure 39-31: IIR_SCTL2 Register Diagram

Table 39-31: IIR_SCTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Software Control Register2.

Secondary Global Control Register

The `IIR_SGCTL` register configures the global parameters for the accelerator in ACM mode for loading CTL1 register as part of TCB.

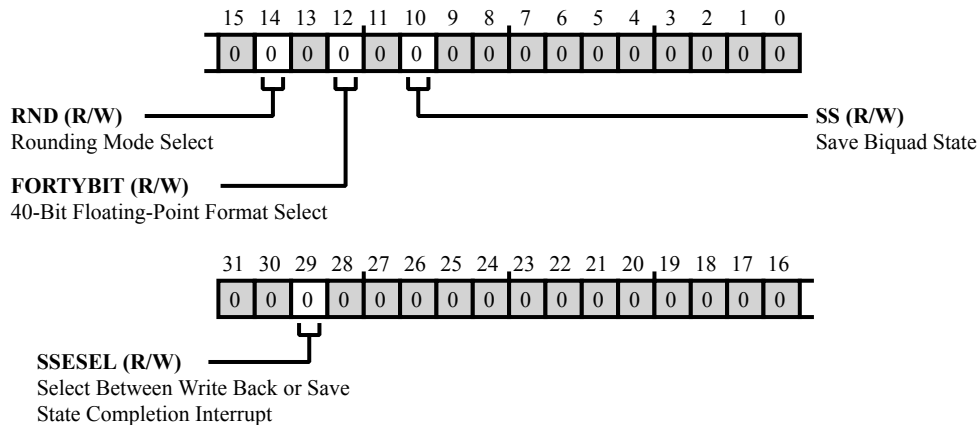


Figure 39-32: IIR_SGCTL Register Diagram

Table 39-32: IIR_SGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	SSESEL	Select Between Write Back or Save State Completion Interrupt.
14 (R/W)	RND	Rounding Mode Select. The <code>IIR_SGCTL.RND</code> bit selects the rounding mode for a floating-point mode compatible with SHARC+.
		0 Round to nearest (even)
		1 Truncate (Round towards zero)
12 (R/W)	FORTYBIT	40-Bit Floating-Point Format Select. The <code>IIR_SGCTL.FORTYBIT</code> bit selects the floating point format.
		0 32-bit floating point
		1 40-bit floating point
10 (R/W)	SS	Save Biquad State. The <code>IIR_SGCTL.SS</code> bit configures the accelerator to store the Dk register settings into the internal memory. This can be used to save the biquad states before switching to another high priority accelerator task.

40 Boot ROM and Booting the Processor

Bootstrapping or booting is the series of events that occur when the system applies power to the processor or when the processor enters a hardware reset state. This section gives an in-depth description of these events and how to integrate an application effectively.

On reset, the processor begins fetching instruction from an internal ROM. The boot code contained within the ROM is designed to facilitate loading an application. The boot code can automatically initialize certain peripherals for communication based on a chosen boot mode, then load an application. For more information on what boot modes are available, see the [Boot Modes](#) section. The boot code can efficiently load an entire application, code, and data, into appropriate locations after the development tools repackage the application into a boot stream.

A boot stream is an application or data that the boot-loader tool splits into blocks. A 16-byte header in each block provides instruction to the boot code for processing the associated data. The processor can perform several boot functions, depending on the flags set in the header. For more details on what options are available and a description of the stream format, refer to the [Boot Loader Stream](#) section.

The boot ROM provides a mechanism through available non-volatile programmable memory (OTP on this processor) to customize different aspects of the boot process. These customizations include: overriding default boot-peripheral instance, overriding default peripheral-timing parameters and disabling boot modes.

NOTE: In this chapter the term boot ROM describes the boot code that cannot be altered. The terms *program* and *application* are used to describe code that is used to customize the boot process.

Several utilities of the boot code are also available to the application. These utilities include features such as copying memory, comparing memory, or loading another boot stream at run time. The APIs may be used to help ensure that application code is more compatible with future products. The boot code also provides the ability to define a custom boot mode. This capability helps when support is not available for a desired boot mode. It allows second stage boot loaders for unsupported boot peripherals to leverage a significant amount of the existing boot ROM functionality.

SRAM Requirements

The boot process reserves 8K bytes of L2 ECC Protected SRAM for dedicated use. This topic describes how the reserved memory region is used during boot.

The boot process requires SRAM resources for stack use and to store various data items that require read/write access during the boot process. 8K bytes of L2 ECC protected SRAM is reserved for this purpose. The *Boot Process SRAM Requirements* table describes the various items stored in this memory region.

Table 40-1: Boot Process SRAM Requirements

Address	Size (Bytes)	Item	Description
0x200FE000	4	Reserved	
0x200FE004	4	Pointer to the struct ADI_ROM_BOOT_CONFIG object	Pointer to the boot configuration structure that is located on the stack. This location is used to find the location of the boot structure on the stack for debug.
0x200FE008	8	Reserved	
0x200FE010	1024	Internal Intermediate Buffer 0	The first of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x200FE410	1024	Internal Intermediate Buffer 1	The second of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x200FE810	16	struct ADI_ROM_BOOT_HEADER object	Storage location for all the block headers of the boot stream.
0x200FE820	2912	Storage for Secure Boot related descriptors	Contains a number of buffers for the various descriptors used by the Cryptographic Accelerators and provides storage for the secure header of a secure boot stream.
0x200FF400	3072	Stack for the boot process	The primary booting cores stack. The processor core should locate the stack in this region in order to preserve security in secure boot operations.

NOTE: To preserve the security of the product, the 8K byte region described here is not a bootable region of memory. If the boot process determines that a block of data in the boot stream is targeted towards this memory region, the boot process terminates and enters either the default error handler or, if applicable, a user-defined error handler. This reserved memory region is free for use after the boot process completes. In order to preserve security when using the boot API to boot a secure boot stream, the stack used during the execution of the boot API must be located at the default location in this reserved 8K bytes region of memory.

Preboot Operations

Preboot is responsible for configuration of all system resources prior to executing the required boot operation.

The steps performed by the preboot process are described here in the order of execution. Numerous stages of the preboot process are conditional based upon the content of the [RCU_BCODE](#) register.

NOTE: When a power on reset, hard reset, and software triggered system reset event completes, the processor operates in PLL Bypass mode (default). Partway through the preboot sequence the processor is brought into Full-On mode at the default settings unless the program has provisioned custom CGU settings in the OTP. The Oscillator Watchdog fault, enabled by default when reset completes, is disabled at this stage unless Oscillator Watchdogs settings are also supplied.

Start-up Sequence

This section describes the initial start-up sequence of all cores in the processor.

When the power-on reset, hardware reset or system reset event completes, the core is released from reset and is responsible for managing the boot process. The following sections describe the sequence of events that occur for the core.

Core Reset Sequencing

When a system and hardware reset event occurs, the processor state is reset and the boot sequence is executed. The core is initially released from reset and starts execution of the preboot software from the boot ROM.

The initial sequence of events after the completion of the reset sequence are shown below.

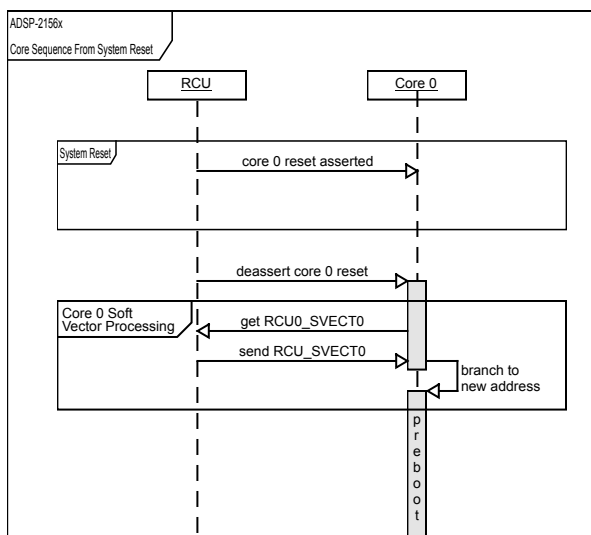


Figure 40-1: Core Reset Sequencing

Core Start-up

Describes the initial operations performed by the SHARC+ core immediately after being released from reset and soft vector processing has completed.

After the core is initially released from the reset event, the soft vectoring process starts the boot process as long as the `RCU_SVECT0` register contained the default reset value during the soft vector processing. The core performs the following initial operations:

- Clear `RCU_MSG.C0IDLE`

- Check for OTPC Boot completion.
- Disable Software Return feature in the BTB.
- Disable the BTB.
- Disable cache via MODE2 register.
- Flush the cache.
- Set IRPTL to 0x00000000.
- Configure primary and secondary DAG configurations to setup the C run-time environment.
- Clear the CBUFEN, SRD1H, SRD1L, SRD2H, SRD2L, ALUSAT, TRUNCATE bits in the MODE1 register.
- Set the IRPTEN, IPERREN, DPERREN, SPERREN bits in the MODE1 register.
- Set MMASK to 0x00000000.
- Set IMASK to 0x00000000.
- Initialize the top 1024 bytes of the stack to initialize the corresponding ECC parity to allow for read access to the initial stack space. This step is bypassed if RCU_BCODE.NOMEMINIT is set.

Table 40-2: Primary/Secondary DAG Configuration for C Run-Time Setup

Primary/Secondary DAG Register	Value	Description
M7, M15	-1	Dedicated registers must always be set to -1
M6, M14	1	Dedicated registers must always be set to 1
M5, M13	0	Dedicated registers must always be set to 0
L0-L5	0	Preserved registers set initially to 0
L6, L7	0	Stack Length Register set initially to 0
L8-L15	0	Preserved registers set initially to 0
B6, B7	0x200FF400	Stack Base Registers
I7	0x200FFFFC	Stack Pointer
L6, L7	0x000002FF	Stack Length Registers
I6	0x200FFFFC	Frame Pointer

Fault Configuration

Describes the initial fault sources that are enabled allowing the processor to signal a fault to the system.

The following faults are enabled via the SEC. Only the faults are enabled, the boot process does not install any SEC interrupts.

Table 40-3: Initial Faults installed during Preboot

SEC Fault ID	SEC Fault Name	Description
0	INTR_SEC0_ERR	SEC Error
3	INTR_WDOG0_EXP	WDOG Expire
4	INTR_WDOG1_EXP	WDOG Expire
5	INTR_OTPC0_ERR	OTPC Expire
9	INTR_L2CTL0_ECC_ERR	L2 ECC Error
32	INTR_SOFT3_INT	Software Driven Interrupt 3. This is raised in the boot ROM error handler should it be entered.
138	INTR_CRC0_ERR	CRC Error
139	INTR_CRC1_ERR	CRC Error
172	INTR_MDMA1_SRC_ERR	MDMA 1 Source DMA Channel Error
173	INTR_MDMA1_DST_ERR	MDMA 1 Destination DMA Channel Error

The SEC is configured for a fault delay of 0x100 via the `SEC_FDLY.COUNT` and `SEC_FSRDLY.COUNT` bit fields. This allows for a delay before the fault assertion is a custom error handler is installed and any SEC interrupts are enabled and handled for more advanced second stage boot scenarios.

The `SYSFAULT` pins are configured via the SEC to support both incoming and outgoing faults by enabling the `SEC_FCTL.FIEN` and `SEC_FCTL.FOEN` bits. The boot process captures an incoming fault if that fault is asserted by the external system on handover to the user application after boot.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed using the `RCU_BCODE.NOFAULTS` bit.

L2 Controller Configuration

The L2 controller is configured to ensure all L2 memory banks are ECC enabled and that the L2 memory scrub feature is disabled.

NOTE: The L2 Controller configuration is enabled by default and may be optionally bypassed by setting the `RCU_BCODE.NOL2CONFIG` bit.

L2 Memory Initialization

The L2 memory subsystem is ECC protected by default. The L2 memory must be initialized to ensure that no read from the L2 memory generates an ECC error. All L2 memory banks are initialized using the `L2CTL_INIT` bit. The boot process waits for each memory bank to signal initialization complete using the `L2CTL_ISTAT` bit. The `RCU_MSG.L2INIT` is set when L2 memory initialization is complete.

NOTE: The L2 memory initialization process is enabled by default and may be optionally bypassed by setting the `RCU_BCODE.NOMEMINIT` bit.

Idle On Entry

The Idle On Entry feature instructs the debugger to halt the boot code before continuing with any further preboot operations.

When this feature is enabled the processor executes a WFI/IDLE instruction and then continues once an event (such as an emulator exception) is serviced.

NOTE: Idle On Entry processing is disabled by default and is enabled by setting the `RCU_BCODE.IDLEONENTRY` bit prior to performing a system reset operation.

SPU Configuration

The SPU is configured differently depending upon the detected security state of the device. The first operation clears any existing security violations that may be indicated in the `SPU_STAT` register.

None of the peripherals are configured to ignore security signals and only the following masters are configured to generate secure transactions.

Table 40-4: SPU Secure Masters during Boot

SPU Endpoint ID	Master Name
110	MDMA0 Source DMA Channel
111	MDMA0 Destination DMA Channel
112	MDMA1 Source DMA Channel
113	MDMA1 Destination DMA channel
108	CRC0
109	CRC1
119	PKTE

SMPU Configuration

The SMPU is used to restrict access to various memory regions in the processor. The configuration applied during boot differs depending on the locked state of the processor.

By default the SMPU instances only allow secure read and write transactions. The *SMPU Configuration* table describes the configurations for the different security states.

Speculative reads are also disabled by setting the `SMPU_CTL.RSDIS` bit.

Table 40-5: SMPU Configuration

SMPU Instance	SMPU Instance	Open <code>SMPU_SECURECTL</code> Value	Locked <code>SMPU_SECURECTL</code> Value
Core_L2_RAM_ROM	2	<code>SMPU_SECURECTL.WNSEN</code> <code>SMPU_SECURECTL.RNSEN</code>	0x00000000 (Default Reset Value)

Table 40-5: SMPU Configuration (Continued)

SMPU Instance	SMPU Instance	Open <code>SMPU_SECURECTL</code> Value	Locked <code>SMPU_SECURECTL</code> Value
DMA_L2_RAM_ROM	3	<code>SMPU_SECURECTL.WNSEN</code> <code>SMPU_SECURECTL.RNSEN</code>	0x00000000 (Default Reset Value)
DMC0 (Only for derivatives with DMC)	9	<code>SMPU_SECURECTL.WNSEN</code> <code>SMPU_SECURECTL.RNSEN</code>	0x00000000 (Default Reset Value)
SPI	11	<code>SMPU_SECURECTL.WNSEN</code> <code>SMPU_SECURECTL.RNSEN</code>	0x00000000 (Default Reset Value)

Secure Debug Key Processing

In the event the processor is locked, the debug tools must submit a secure debug key that is matched with a key on the processor. A 128-bit Secure Debug Key must be provided the application program prior to locking the device.

The secure debug key is read from the OTP memory and then written to the corresponding register in the TAPC. After the key has been written the `TAPC_SDBGKEY_CTL.VALID` bit is set. Once the debug tools then submit their key a key compare operation is performed.

Debug tools must wait for the boot software to load the key before setting the `TAPC_SDBGKEY_CTL.VALID` bit before submitting the key for comparison.

The 128-bit Secure Debug Key is loaded as follows from the storage area in OTP.

Table 40-6: Secure Debug Key Load Procedure

Secure Debug Key[127:0]	Register
Secure Debug Key[31:0]	<code>TAPC_SDBGKEY0</code>
Secure Debug Key[63:32]	<code>TAPC_SDBGKEY1</code>
Secure Debug Key[95:64]	<code>TAPC_SDBGKEY2</code>
Secure Debug Key[127:96]	<code>TAPC_SDBGKEY3</code>

CAUTION: A 16 bit non-zero value programmed on the `emu_key_disable` field in OTP or a key of 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF programmed in the secure debug key field provided in OTP results in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by the program. If the processor fails to boot (due to corrupted firmware) then there is no debug access. The only way to gain access is to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger.

CGU Configuration

This step in the preboot process reconfigures the internal clocks on the processor for improved boot performance.

The boot process can optionally configure the CGU in order to improve boot performance. The settings for the CGU are located within the `struct ADI_ROM_OTP_BOOT_CGU_INFO` structure that has storage allocated in the OTP as part of the `struct ADI_ROM_OTP_BOOT_INFO` structure.

Typically, CGU configuration is performed using an **Init Block** in the boot stream. This provides greatest flexibility. In situations where boot time must be kept to a minimum, settings in the OTP can be applied at this stage of pre-boot as opposed to during the boot process itself. When a processor is locked, the boot process does not support an **Init Block** in the boot stream. For a locked processor programs must use the OTP to reconfigure the clocks without adopting a multi-stage boot strategy.

On release from reset, the CGU is configured for PLL Bypass mode. To improve boot performance, the boot software reconfigures the CGU so that Full-On mode is entered with the default CGU settings.

If the program doesn't provide settings in OTP to configure the Oscillator Watchdog, then the Oscillator Watchdog fault, enabled by default after reset, is disabled prior to reconfiguring the CGU.

If settings are provided in the OTP to configure the Oscillator Watchdog, the fault is left enabled and, if applicable, the CGU is configured per the provided settings. The remainder of the boot process completes in Full-On mode.

The CGU configuration object (see `struct ADI_ROM_OTP_BOOT_CGU_INFO`) allows the boot software to configure the CGU for a more efficient boot process.

Table 40-7: ADI_ROM_OTP_BOOT_CGU_INFO Members

Type	Name	Description
uint32_t	ctl_WEN:1 (bitfield)	Enable write to the <code>CGU_CTL</code> register
uint32_t	div_WEN:1 (bitfield)	Enable write to the <code>CGU_DIV</code> register
uint32_t	Reserved0:1 (bitfield)	Reserved
uint32_t	div_DSEL:5 (bitfield)	<code>CGU_DIV.DSEL</code> value
uint32_t	div_CSEL:5 (bitfield)	<code>CGU_DIV.CSEL</code> value
uint32_t	div_S0SEL:3 (bitfield)	<code>CGU_DIV.S0SEL</code> value
uint32_t	div_SYSSEL:5 (bitfield)	<code>CGU_DIV.SYSSEL</code> value
uint32_t	div_S1SEL:3 (bitfield)	<code>CGU_DIV.S1SEL</code> value
uint32_t	div_OSEL:7 (bitfield)	<code>CGU_DIV.OSEL</code> value
uint32_t	ctl_DF:1 (bitfield)	<code>CGU_CTL.DF</code> value
uint32_t	ctl_MSEL:7 (bitfield)	<code>CGU_CTL.MSEL</code> value
uint32_t	auto_disable:1 (bitfield)	disable polling on auto-alignment of clocks (not recommended)
uint32_t	Reserved1:6 (bitfield)	Reserved
uint32_t	clkoutsel_CLKOUTSEL:5 (bitfield)	<code>CGU_CLKOUTSEL.CLKOUTSEL</code> value
uint32_t	clkoutsel_WEN:1 (bitfield)	Enable write to the <code>CGU_CLKOUTSEL</code> register

Table 40-7: ADI_ROM_OTP_BOOT_CGU_INFO Members (Continued)

Type	Name	Description
uint32_t	Reserved2:12 (bitfield)	Reserved
uint32_t	oswctl0_WEN:1 (bitfield)	Enable write to the CGU_OSCWDCTL instance 0 register
uint32_t	oswctl0_HODF:6 (bitfield)	CGU_OSCWDCTL.HODF value
uint32_t	oswctl0_HODEN:1 (bitfield)	CGU_OSCWDCTL.HODEN value
uint32_t	oswctl0_CNGEN:1 (bitfield)	CGU_OSCWDCTL.CNGEN value
uint32_t	oswctl0_BOUF:5 (bitfield)	CGU_OSCWDCTL.BOUF value
uint32_t	oswctl0_BOUEN:1 (bitfield)	CGU_OSCWDCTL.BOUEN value
uint32_t	oswctl0_FAULTEN:1 (bitfield)	CGU_OSCWDCTL.FAULTEN value
uint32_t	oswctl0_MONDIS:1 (bitfield)	CGU_OSCWDCTL.MONDIS value
uint32_t	oswctl0_FAULTPINDIS:1 (bitfield)	CGU_OSCWDCTL.FAULTPINDIS value
uint32_t	Reserved3:14 (bitfield)	Reserved

If a CGU_STAT.WDIVERR, CGU_STAT.WDFMSERR, CGU_STAT.LWERR or CGU_STAT.ADDRERR occurs at the entry to or completion of the configuration routine, the default error handler is called and the boot process terminates.

NOTE: Programs can bypass CGU configuration by setting the RCU_BCODE.NOPREBOOT bit when this part of the boot process is reached.

L1 Memory Initialization

The processor initializes all parity and ECC protected memories to perform subsequent read operations without generating an ECC or parity error.

The *L1 Memory Initialization* table describes the methods used to initialize the various parity and ECC supported memories on the processor.

Table 40-8: L1 Memory Initialization

Resource To Fill Memory	Memory Type	Address	Count	Fill Value	Flag Set Upon Completion
Core 0	Core 0 L1 Bank 0	0x00048000	0x6000 (LW)	0x00000000	RCU_MSG.COL1INIT
Core 0	Core 0 L1 Bank 1	0x00058000	0x6000 (LW)	0x00000000	
Core 0	Core 0 L1 Bank 2	0x00060000	0x4000 (LW)	0x00000000	
Core 0	Core 0 L1 Bank 3	0x00070000	0x4000 (LW)	0x00000000	

NOTE: The Memory Initialization process is enabled by default and may be optionally bypassed by setting the RCU_BCODE.NOMEMINIT bit.

Default Entry Point

The first instruction executed for the core in the boot ROM is a read of the `RCU_SVECT0` register. The boot ROM then vectors to that location.

The *Default Entry Point* table defines the default entry point for core in the processor

Table 40-9: Default Entry Point

Core ID	Register	Entry Point
0	<code>RCU_SVECT0</code>	0x00500004

The boot code does not set any initial default application entry points by writing the entry point to the `RCU_SVECT0` register in the processor. It will hold the default values of the power on reset case. The `RCU_SVECT0` register is updated at the end of the boot process. For the secure boot, the register is updated only after the successful authentication of the boot stream.

NO-BOOT Processing

No-Boot mode is executed when selected by the `SYS_BMODE[n]` pins. The boot mode is intended as a recovery boot mode or for debug purposes. The core simply executes in an endless loop in the boot ROM, terminating further execution of the boot process.

This boot mode is primarily intended for debug sessions when no boot source may be configured. It allows for a debugger to safely connect to the device and take control (assuming the debugger has been granted access rights as defined by the processor security implementation).

NOTE: NO-BOOT processing is usually only entered as a result of the boot mode pin sampling resulting in execution of the No-Boot boot mode. This processing can also be optionally enabled by setting `RCU_BCODE.HALT` field. The `RCU_BCODE.HALT` setting can be especially useful for debug sessions to force the execution of the NO-BOOT mode regardless of the `SYS_BMODE[n]` state allowing a user application to be loaded via the debug tools without fear of the image being corrupted as a result of attempting to boot through another source.

SYS_RESOUT Signal

In order to signal to the external system that the processor is in a configured state and ready to start the boot process, the boot software de-asserts the `SYS_RESOUT` pin via the `RCU_CTL.RSTOUTDSRT` bit.

DMC Configuration

To boot to external DDR memories (to support booting to external memory when the processor is locked) the DMC must be configured.

Typically, DMC configuration is done using the [Init Block](#) in the boot stream. When the processor is locked the boot process does not support an [Init Block](#) in the boot stream. For a locked processor programs must use the OTP to configure the DMC without adopting a multi-stage boot strategy.

Table 40-60 [ADI_ROM_OTP_DMC_CONFIG Members](#) provides details of the OTP region that is used to store the DMC configuration. In the table, the `ADI_ROM_OTP_BOOT_CFG::dmcEn` must be set in order for the settings to apply.

Table 40-10: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
uint32_t	ulDDR_DLLCTLCFG:16 (bit field)	Content of DMC CFG[15:0]
uint32_t	ulDDR_DLLCTLCFG:16 (bit field)	Content of DMC DLL CTL[15:0]
uint32_t	ulDDR_EMR2EMR3:16 (bit field)	Contents of DMC_EMR3 [15:0]
uint32_t	ulDDR_EMR2EMR3:16 (bit field)	Contents of DMC_EMR2 [15:0]
uint32_t	ulDDR_CTL	Content of DMC CTL[31:0]
uint32_t	ulDDR_MREMR1:16 (bit field)	Content of DMC EMR1[15:0]
uint32_t	ulDDR_MREMR1:16 (bit field)	Content of DMC MR[15:0]
uint32_t	ulDDR_TR0	Content of DMC_TR0[31:0]
uint32_t	ulDDR_TR1	Content of DMC_TR1[31:0]
uint32_t	ulDDR_TR2	Content of DMC_TR2[31:0]
uint32_t	ulDDR_ZQCTL0	Content of DMC PHY ZQCTL0[31:0]
uint32_t	ulDDR_ZQCTL1	Content of DMC PHY ZQCTL1[31:0]
uint32_t	ulDDR_ZQCTL2	Content of DMC PHY ZQCTL2[31:0]
uint32_t	ulDDDRPHY_CACTL	Content of DMC PHY CA_CTL [31:0]
uint32_t	uBypassDelay_LANE0CTL1:6 (bit field)	Content of DMC LANE0_CTL1[15:10]
uint32_t	uBypassDelay_LANE1CTL1:6 (bit field)	Content of DMC LANE1_CTL1[15:10]
uint32_t	uBypassDelay_LANE0CTL0:6 (bit field)	Content of DMC LANE0_CTL0[15:10]
uint32_t	uBypassDelay_LANE1CTL0:6 (bit field)	Content of DMC LANE1_CTL0[15:10]
uint32_t	reserved0:8 (bit field)	Reserved

There is an additional single bit located in OTP, `ADI_ROM_OTP_BOOT_CFG::dmcInv` allowing programs to invalidate the DMC settings stored in OTP.

NOTE: Once `ADI_ROM_OTP_BOOT_CFG::dmcInv` has been set in OTP there is no means to configure the DMC during the preboot phase.

The configuration of the DMC is bypassed if the `RCU_BCODE.NOPREBOOT` bit is set when this part of the boot process is reached.

Bypassing the Boot Process

The boot process can be bypassed allowing a program to start execution from the address stored in the core's soft vector register. This is useful when working in emulation sessions as it provides a mechanism to be able to execute programs directly from an accessible memory that already contains executable code.

To bypass the boot process, set the `RCU_BCODE.NOKERNEL` bit. The processor core vectors to the address stored in the core's corresponding `RCU_SVECT0` register instead of calling the required boot mode.

This feature is used along with other features such as disabling memory initialization.

Boot Mode Disable

Specific boot modes can be permanently disabled via OTP. If the disabled boot mode is enabled using the `SYS_BMODE[n]` pins, a boot error is generated.

A byte of storage is provided in the OTP to disable up to eight boot modes. The boot mode disable field can be programmed using the `adi_rom_otp_pgm()` routine. The `otp_data::bootModeDisable` member is used in the program operation to disable the various boot modes.

Table 40-11: Boot Mode Disable

<code>otp_data::bootModeDisable</code> Bit Position	Corresponding Boot Mode
0	SPI Master Boot Mode
1	SPI Slave Boot Mode
2	UART Slave Boot Mode
3	Linkport Slave Boot Mode
4	OSPI Master Boot Mode
7-5	Reserved

Boot Command Customization

Boot command customization allows permanent customization of a particular boot mode. For example, it is possible to change the peripheral instance used for boot operation.

Storage is provided in OTP for a command item for each supported boot mode.

Storage is provided in the `struct ADI_ROM_OTP_BOOT_CMD_INFO` member of `struct ADI_ROM_OTP_BOOT_INFO`. Refer to the corresponding boot modes boot command description for details on supported command options.

NOTE: Before programming boot command to the OTP, evaluate the boot command using the `adi_rom_Boot()` API and ensure that the boot command provides the desired functionality. Once the command is programmed to OTP, it cannot be reverted to original default settings.

Boot Mode Specific SPU Configuration

Prior to performing actual boot process, the processors SPU resources specific to the boot mode selected are configured. This is performed in the preboot phase as opposed to within the boot mode itself when calling the boot API, as it isolates the security functionality of the processor allowing it to be handled specifically by a separate process.

The following additional SPU resources are configured as secure masters according to the boot mode selected.

Table 40-12: Boot Mode Specific SPU Configuration

Boot Mode	SPU Endpoint ID	Master Name
SPI Master Boot (Memory-Mapped Mode) and OSPI Master Boot	110, 111	MDMA0 Source DMA Channel, MDMA0 Destination DMA Channel
SPI Master Boot (Peripheral Mode)	78, 76, 74	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
SPI Slave Boot	78, 76, 74	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
UART Slave Boot	65, 67, 69	UART 0 Receive DMA, UART1 Receive DMA, UART2 Receive DMA
LINKPORT Slave Boot	5, 6	LINKPORT 0 DMA, LINKPORT 1 DMA

NOTE: In the *Boot Mode Specific SPU Configuration* table, for a given boot mode, all the SPU resources are not configured. Only a single peripheral instance is enabled according the peripheral instance selected for boot. For example if the boot command for the boot mode indicates boot from UART0 only the UART0 Receive DMA is configured, the other UART Receive DMAs are not configured for secure access.

Executing the Boot Mode

The boot mode is called `adi_rom_Boot()` routine. The routine fetches and processes the boot stream from the configured boot source.

The table provides default parameters passed to each of the supported boot modes. For details on the API usage, refer to `adi_rom_Boot()`.

Table 40-13: Default Boot ROM API Parameters

Boot Mode	pAddress	flags	blockCount	pHook	Command
No Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000000
SPI Master Boot	0x60000000	0x00040000	0x00000000	Points to an empty routine in ROM	0x0000207
SPI Slave Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x0000212
UART Slave Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x0000013

Table 40-13: Default Boot ROM API Parameters (Continued)

Boot Mode	pAddress	flags	blockCount	pHook	Command
LINKPORT Slave Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000014
OSPI Master Boot	0x60000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000308
Reserved	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000000
Reserved	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000000

The hook function installed via pHook on this product does not perform any additional configuration.

Boot Modes

The boot implementation provides built-in support for booting from various peripherals.

The *Booting Modes* table describes the supported boot modes.

In slave boot mode, the processor functions as a slave to any host device. In these modes, the host device usually controls the processor SYS_HWRST input. Typically, the host applies the reset sequence and waits until the processor is ready to boot, depending on the peripheral in use, and transmits the boot stream data to the processor. Handshake signals are used to signal to the host that the processor is ready to accept more data.

In master boot mode, the processor controls the peripheral and requests data via the peripheral as and when required.

Individual boot modes can be disabled. For more information about disabling boot modes, see [Boot ROM OTP Customizations](#).

Table 40-14: Booting Modes

SYS_BMODE[2:0]	Boot Source	Description
000	No Boot	The processor does not boot. Rather the boot kernel executes some of the preboot operations then enters an endless WFI/IDLE state.
001	SPI Master Boot	Boot from integrated Flash memory through the SPI2 peripheral configured for memory-mapped mode.
010	SPI Slave Boot	Boot through the SPI2 peripheral configured as a slave.
011	UART Boot	Boots through UART0 configured as a slave receiver.
100	LINKPORT Boot	Boot through LINKPORT0 peripheral configured as a slave receiver. This boot mode is only applicable to derivatives supporting the LP0 instance.
101	OSPI Master Boot	Boot from integrated Flash memory through the OSPI peripheral configured for memory-mapped mode.

Table 40-14: Booting Modes (Continued)

SYS_BMODE[2:0]	Boot Source	Description
110, 111	Reserved	

No-Boot Mode

No-Boot mode is intended for device recovery purposes caused due to incorrect programming of the boot source memory allowing for target connection through an emulator. Emulation tools can also leverage the No-Boot functionality allowing for debug sessions to run the preboot software prior to loading an application while preventing the boot process from continuing and clobbering data loaded by the emulator.

This boot mode results in several preboot operations being performed before placing the core in a safe, endless loop located in the boot ROM.

For a complete list of operations performed when No-Boot is selected, see [Preboot Operations](#). The core terminates at the [NO-BOOT Processing](#) stage of the preboot process.

SPI Master Boot Mode

The SPI master boot routine provides support for booting the processor from SPI flash memories. The SPI boot mode uses a device auto-detection feature that is enabled by default. This lets the boot stream itself instruct updates to the SPI configuration and the read command used allowing for more efficient transactions.

NOTE: For default SPI master mode, the peripheral fault, SPI2_ERR with Fault ID 72 is enabled.

Boot From External SPI Flash Devices

The SPI boot mode supports booting from 24-bit or 32-bit addressable flash devices. The boot mode uses the MDMA channels by default and configures the SPI flash for memory-mapped functionality. Peripheral DMA mode is also supported when calling the boot mode via `adi_rom_Boot()`.

When auto-device detection is enabled, the SPI memory is initially read using the standard 0x03 SPI read command with a reduced clocking frequency for maximum compatibility. The first nibble of the boot stream is then used to reconfigure the SPI interface and possible the SPI flash. Refer to [SPI Device Detection Routine](#).

NOTE: Support for automatic device detection via the first nibble of the boot stream is not supported when booting secure boot streams. Instead when signing the boot image an attribute can be set in the image header that specifies the configuration to use.

For booting, the SPI memory is connected as shown in the *SPI Memory Connections* figure.

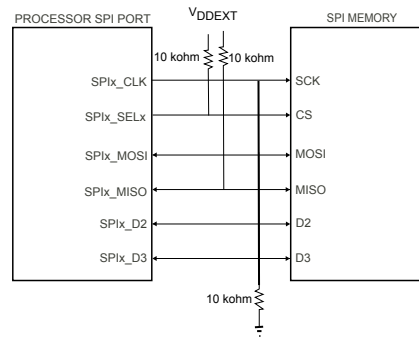


Figure 40-2: SPI Memory Connections

The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in a high-impedance mode such as during reset.

Initialization codes are allowed to manipulate the `ADI_ROM_BOOT_CONFIG::dBootCommand` to extend boot mechanism to a second SPI memory connected to another slave select pin. Updating the field that specifies the slave select signal for use, allows the boot process to manage larger boot streams than fit into a single SPI device.

NOTE: If modifying the slave select signal used during the boot process, configure the pin multiplexing to enable the correct functionality for the pin. Once the boot process has proceeded past the configuration function and the boot process has actually started, the boot kernel will not perform any further pin multiplexing operations.

SPI Device Detection Routine

Since the boot mode supports booting from various SPI memories, the boot kernel automatically detects what type of memory is connected. To determine whether the SPI memory device requires a 24-bit or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The `SPI_MISO` signal requires a pull-up resistor. The routine relies on the fact that memories do not drive their data outputs unless the right number of address bytes are received.

Initially, the boot kernel transmits the read command on the `SPI_MOSI` line. Once the command has been sent, the boot kernel proceeds to transmit a single address byte and waits until the receive FIFO indicates that the buffer is no longer empty. The first received byte is discarded. The boot code then proceeds to issue another address byte while simultaneously receiving a byte. The process continues until a non `0xFF` or `0x00` byte is received or until the full 4 address bytes is sent without any valid data being returned.

The receiving of a non `0x00` or `0xFF` byte tells the boot code whether the memory device requires 24 or 32 address bits. The lower nibble of the received byte is then used to further customize the boot mode. This nibble is referred to as the *BCODE*. The boot code applies settings to the SPI peripheral according to the *SPI Master Boot BCODE Descriptions*.

If the received value equals `0x00` or `0xFF`, it is assumed that the memory device has not driven its data output thus, another zero byte is transmitted and the received data is tested again.

If the value still equals 0xFF, device detection continues. Device detection aborts immediately when a byte different than 0xFF is received. The boot process continues with normal boot operation and it reissues a command to again read from address 0. Two read sequences load the first block header. Separate read sequences load further block headers and block payload fields.

The *SPI Device Detection Principle* figure illustrates how individual devices behave.

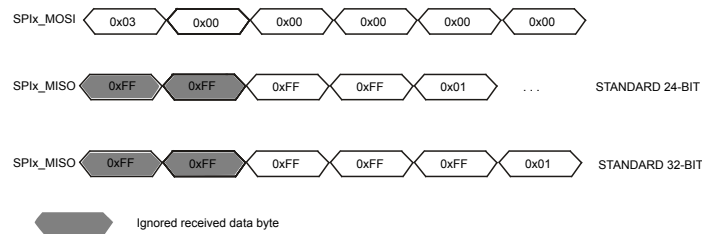


Figure 40-3: SPI Device Detection Principle

Table 40-15: SPI Master BCODE Configuration Lookup Table

Member	BCODE				
	0	1	2	3	4
Transfer Type	Single bit command, address and data	Single bit command, address and data	Single bit command, address and data	Single bit command, address and dual bit data	Single bit command, address and Quad data
ubDummyBytes	0x00	0x00	0x01	0x01	0x01
ubReadCommand	0x03	0x03	0x0B	0x3B	0x6B
ubDataBits	0x00	0x00	0x00	0x01	0x02
ubAddressBytes	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x000F	0x000F	0x0001	0x0001	0x0001
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000
nTxCtl	0x00000003	0x00000003	0x00040003	0x00040033	0x00040033
nRxCtl	0x00000003	0x00000003	0x00040003	0x00140003	0x00240033
nCmdCtl	0x00000003	0x00000003	0x00000003	0x00000033	0x00000033
pMIOEnFunction	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00

Table 40-16: SPI Master BCODE Configuration Lookup Table

Member	BCODE					
	5	6	7	8	9	A
Transfer Type	Single bit command, Dual bit address and data	Single bit command, Dual bit address and data	Single bit command, Dual bit address and data	Single bit command, Quad bit address and data	Single bit command, Quad bit address and data	Single bit command, Quad bit address and data

Table 40-16: SPI Master BCODE Configuration Lookup Table (Continued)

Member	BCODE					
	5	6	7	8	9	A
ubDummyBytes	0x01	0x02	0x03	0x02	0x03	0x05
ubReadCommand	0xBB	0xBB	0xBB	0xEB	0xEB	0xEB
ubDataBits	0x01	0x01	0x01	0x02	0x02	0x02
ubAddressBytes	0x03	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
nTxCtl	0x00140033	0x00140033	0x00140033	0x00240033	0x00240033	0x00240033
nRxCtl	0x00140033	0x00140033	0x00140033	0x00240033	0x00240033	0x00240033
nCmdCtl	0x00000033	0x00000033	0x00000033	0x00000033	0x00000033	0x00000033
pMIOEnFunction	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00	0x00

NOTE: For the above configurations, the addressing scheme can be 3-bytes or 4-bytes depending on the addressing of flash detected in auto-detection. The SPI mode byte issued for all the SPI master peripheral based configurations is 0x00. The mode byte is the first byte transmitted after the address cycles and is used to control the continuous read mode functionality in which the next read operation is not required to issue a command cycle. Continuous read mode is not supported during the boot process.

Supported Quad Mode Enable Methods

The boot ROM does not support enabling quad mode on the SPI flash device. To boot in quad mode, the flash device must be configured outside the boot ROM.

NOTE: It is more beneficial to boot initially in dualmode and use an initcode to enable Quad mode.

Run-Time API

The following table provides descriptions of the [adi_rom_Boot\(\)](#) command parameter.

Table 40-17: SPI Master Boot Command Descriptions

Bits	Name	Setting	Description
31:28	ROM_BCMD_SPIM_SPEED	0 to 1111	SPI clock divider. This value written to the SPI peripherals clock divider register
27	Reserved	Reserved	Reserved

Table 40-17: SPI Master Boot Command Descriptions (Continued)

Bits	Name	Setting	Description
26:22	ROM_BCMD_SPIM_DUMMY	0000	No dummy bytes required after the address
		00001	1 dummy byte required after the address
	
		11111	31 dummy bytes required after the address
21:20	ROM_BCMD_SPIM_ADDR	00	Flash device requires an 8-bit address
		01	Flash device requires a 16-bit address
		10	Flash device requires a 24-bit address
		11	Flash device requires a 32-bit address
19:16	ROM_BCMD_SPIM_BCODE	0000 to 1111	Boot mode-specific code. Specifies the boot mode-specific code that can further customize and control the boot process.
15	ROM_BCMD_SPIM_CMD	0	Issue the read command over the single bit bus
		1	Issue the read command over the multi-bit bus. Not recommended for use in ADSP-2156x processor.
14:12	ROM_BCMD_SPIM_SSEL	000	Use slave select 1 for SPI chip select
		001	Use slave select 2 for SPI chip select
		010	Use slave select 3 for SPI chip select
		011	Reserved
		100	Reserved
		101	Reserved
		110	Reserved
		111	Reserved

Table 40-17: SPI Master Boot Command Descriptions (Continued)

Bits	Name	Setting	Description
11:8	ROM_BCMD_SPIM_DEVE- NUM	0 to 16	Boot peripheral enumeration. So for SPI2 it would be set to 2, SPI0 it is set to 0. 0x3 to 0xF is reserved
7	Reserved	Reserved	Reserved
6	ROM_BCMD_SPIM_NOAU- TO	0	Automatic device detection disable. Perform automatic device detection and peripheral configuration based on the BCODE value (first nibble) of the boot streams block header
		1	Do not perform automatic device detection
5	ROM_BCMD_SPIM_NOCFG	0	Device configuration enable. Instructs the config routine to perform pinmuxing configuration and full peripheral configuration
		1	Device configuration disable.
4	ROM_BCMD_SPIM_HOST	0	Master boot mode enable.
		1	Slave boot mode enable.
3:0	ROM_BCMD_SPIM_DEVICE	0000	Reserved
		0001	Reserved
		0010	SPI Boot (Legacy peripheral DMA)
		0011	Reserved
		0100	Reserved
		0101	Reserved
		0110	Reserved
		0111	SPI Memory Mapped Boot
		1000 to 1111	Reserved

NOTE: All bits in the above table that are not defined must be set to zero. Supported features may be limited depending on peripheral instance.

SPI Slave Boot Mode

When using SPI slave mode boot, the processor consumes boot data from an external SPI host device. This mode supports single, dual, and quad-bit modes. The boot kernel always starts in single bit mode and can be changed

using the appropriate command. The following figures show the hardware configuration for the modes. As in all slave boot modes, the host device controls the SYS_HWRST input of the processor.

NOTE: Secure Boot Stream Padding

For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

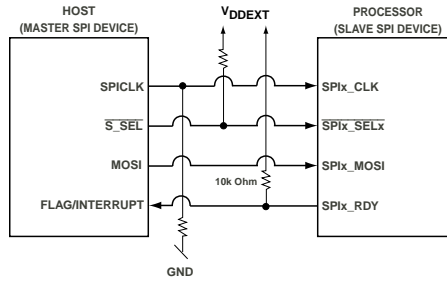


Figure 40-4: Connection between Host (SPI Master) and Processor (SPI Slave)

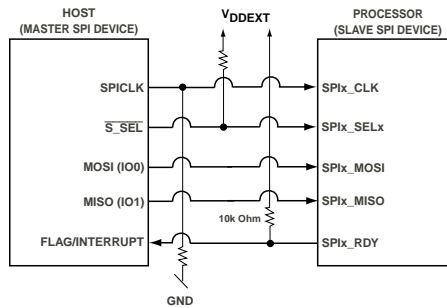


Figure 40-5: Connection between Host (SPI Master) and Processor (SPI Slave) DIOM

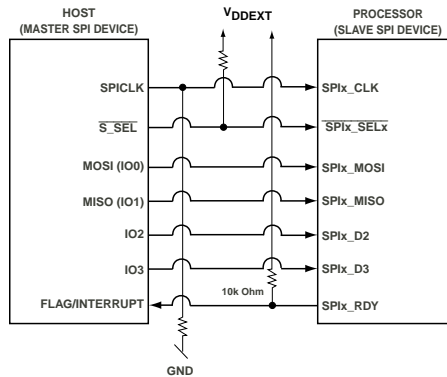


Figure 40-6: Connection between Host (SPI Master) and Processor (SPI Slave) QSPI

The host drives the SPI clock and is responsible for timing. The host must provide an active-low chip select signal that connects to the processor's SPIx_SS input signal with each byte transferred or remain low during the entire procedure. 8-bit data is expected and 16-bit mode is not supported.

In SPI slave boot mode, the boot kernel sets the SPI_CTL.CPHA bit and clears the SPI_CTL.CPOL bit. Therefore the SPI_MISO pin is latched on the falling edge of the SPI_MOSI pin.

The SPI slave processor detects the correct boot mode from the host SPI device by reading the first byte sent, defined as *SPICMD*. The *SPICMD Descriptions* table describes the available codes. These additional bytes must be sent prior to transmitting the data to configure the SPI device.

The *SPICMD Descriptions* table describes the following:

- Host starting in single bit mode
- Host starting in a mode other than single bit

Table 40-18: SPICMD Descriptions

SPICMD	Description
<i>If host starts in Single bit Mode</i>	
0x3	Keep single-bit mode
0x7	Switch to dual-bit mode
0xB	Switch to quad-bit mode
<i>If host device starts in DIOM or QSPI</i>	
0xAA,0xBF	Switch to dual-bit mode
0xEE,0xEE,0xFE,0xFF	Switch to quad-bit mode

In SPI slave boot mode, *SPIx_RDY* functionality is critical. The *SPIx_RDY* output is used for back pressure and requires a pulling resistor. The boot code requires the *SPIx_RDY* signal function as active-low. The host is only permitted to transfer data when *SPIx_RDY* is in the active state. This functionality allows the processor to hold off the host while the processor is in reset or executing the pre-boot and processor initialization sequences. The SPI is configured to deassert *SPIx_RDY* when the receive FIFO is filled to 75% or more.

NOTE: For default SPI slave mode, the peripheral fault, *SPI2_ERR* with Fault ID 72 is enabled.

The *SPI Program Flow on the Host Side* figure illustrates the required program flow on the host side.

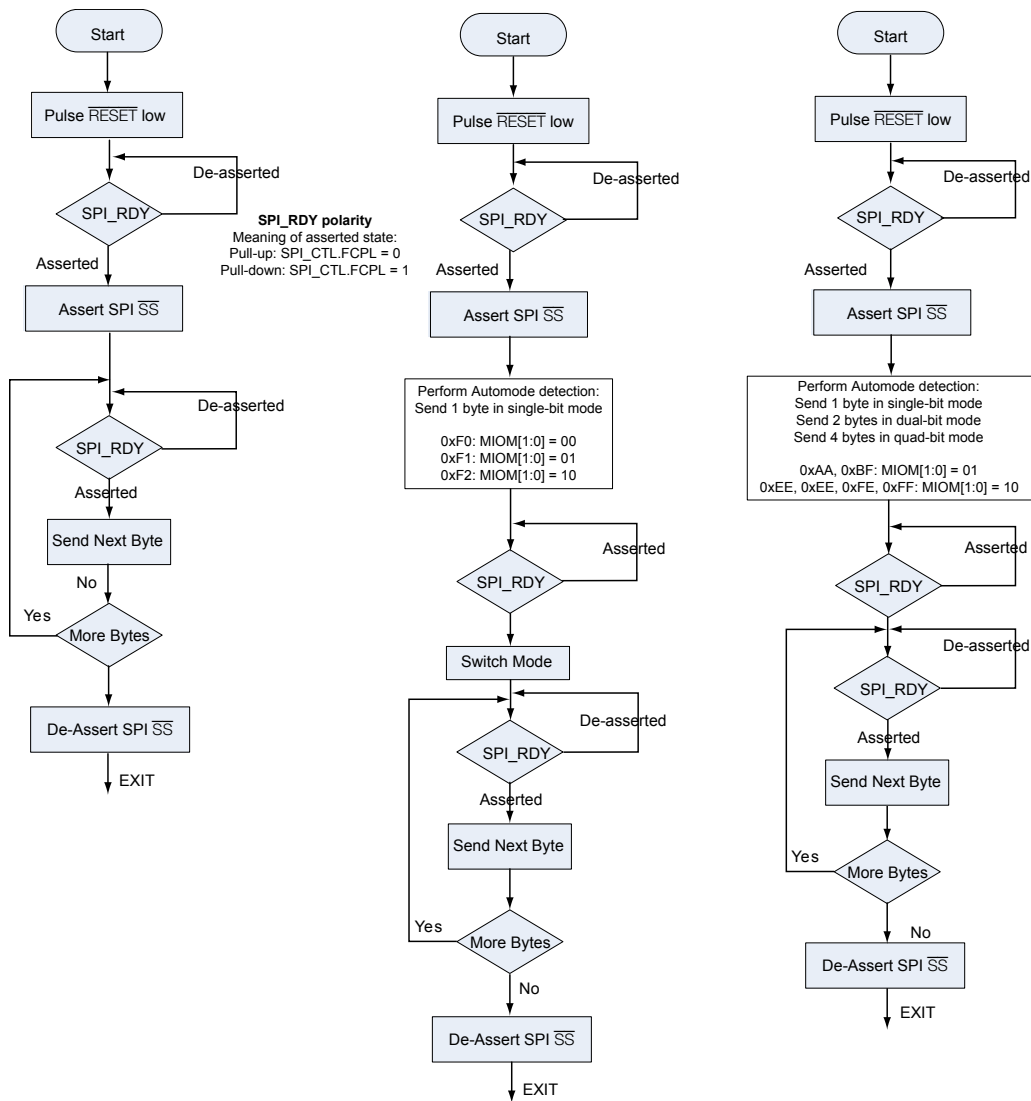


Figure 40-7: SPI Program Flow on the Host Side

Run-time API

The SPI slave boot mode can be called through the Boot Routine API function at run-time. Initiating a boot through the run-time API allows for extra customization such as disabling automatic device configuration or specifying a different SPI device other than the default.

When ROM_BCMD_NOCFG flag is specified, it is necessary to program pin multiplexing and other SPI configuration as required, while keeping the SPI_CTL.EN bit cleared.

The ROM_BCMD_NOAUTO flag can suppress auto mode detection. In that case, the desired configuration must be passed through the ROM_BCMD_SPIS_BCODE bit field, even if the ROM_BCMD_NOCFG flag is set.

The following table provides descriptions of the adi_rom_boot parameter.

Table 40-19: SPI Slave Boot Command Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration
19:16	ROM_BCMD_S PIS_BCODE	Boot Mode Specific BCODE. Specifies the boot mode-specific code that can further customize and control the boot process.
		00xxb Single bit SPI bus
		01xxb Dual SPI bus
		10xxb Quad SPI bus
		11xxb Reserved
11:8	ROM_BCMD_ DEVENUM	Device enumeration. Specifies the SPI device to use.
		0x0 SPI0
		0x1 SPI1
		0x2 SPI2
		0x3 - 0xF Reserved
6	ROM_BCMD_ NOAUTO	Automatic device detection disable. When set, this bit disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.
5	ROM_BCMD_ NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.
4	ROM_BCMD_ HOST	Host boot mode enable. When set, enables SPI slave boot mode. Otherwise, use the master boot mode.
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.
		0x2 SPI

NOTE: All bits in the table that are not defined must be set to zero. Supported features may be limited depending on peripheral instance.

Link Port Slave Boot Mode

This section describes booting from the link port with the processor as a slave.

Link port boot is a slave boot mode in which the processor receives boot data from an external link port master through link port 0. The link port is configured for receive mode and all transfers from the link port to memory are performed under the control of DMA. The maximum supported operating frequency of the link port is 66 MHz for which the master boot source is responsible for deriving the clock frequency. The link port receiver operates at an asynchronous frequency up to the maximum supported operating frequency.

The link port protocol supports a way to generate link port transmit and receive service requests. The transmit service request is generated on the processor to transmit data when the transmitter is disabled. The receiver drives the LACK \times signal high to initiate this activity. The receive service request is generated on a receiver when it is disabled. The transmitter drives the LCLK \times signal high to initiate this activity.

Because the transmitter and receivers can be enabled at different times, external pull-down resistors are required on both the LCLK \times and LACK \times signals to eliminate any false service request assertions.

The link port slave boot mode initialization phase waits for the receive service request before passing control back to the main kernel. Once this initial receive service request has been detected, the receiving link port is enabled and the boot process completes. The receiving link port is not disabled again until after boot is complete. Once the link port is enabled, the receive DMA channel controls all transfers. The load function for the link port receive boot mode can then point to the peripheral DMA routine of the main kernel in a similar way to the SPI slave boot mode.

NOTE: For default link port slave mode, the peripheral DMA fault, LP0_DMA_ERR with Fault ID 168 is enabled.

Run-time API

The linkport slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both device auto-detection and device configuration can be disabled, and a device other than the default LINKPORT0 can be specified.

If ROM_BCMD_NOCFG flag is specified, it is the programs responsibility to configure pin multiplexing as required.

The following table provides descriptions of the adi_rom_Boot parameter.

Table 40-20: LINKPORT Slave Boot command Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the LINKPORT device to use.	
		0x0	LP0
		0x1	LP1
		0x2 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, enables SPI slave boot. Otherwise, use the master boot mode.	

Table 40-20: LINKPORT Slave Boot command Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.
		0x4 LINKPORT

NOTE: All bits in the above table that are not defined must be set to zero. Supported features may be limited depending on peripheral instance.

UART Slave Boot Mode

When using UART slave mode boot, the processor receives boot data from a UART host device connected to the UART interface. The device connected to UART0 is initially detected using an autobaud detection sequence. After finishing the UART slave boot process, all control and status registers of the used resources are restored.

Further customization, such as disabling autobaud detection, and changing the device, use the boot routine API.

During boot operation, the host device usually relies on the RTS output of the UART device. At boot time, the processor does not evaluate RTS signals driven by the host. Since the RTS is in a high impedance state when the processor is in reset, or while executing a pre-boot, an external pull-up resistor to VDDEXT is recommended. The *Connection Between Host and Processor* figure shows the interconnection required for booting. The figure does not show physical line drivers and level shifters that are typically required to meet the individual UART-compatible standards.

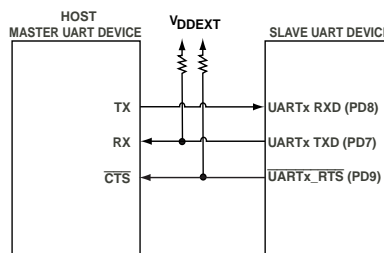


Figure 40-8: Connection Between Host and Processor

When the UART is enabled, the RTS immediately transitions low, encouraging the host to send the first boot stream data as shown in the *Host Relying on RTS* figure. For half-duplex UART connections, the host must avoid this action. The host must wait until it has received the 4 bytes from the slave processor before sending any data.

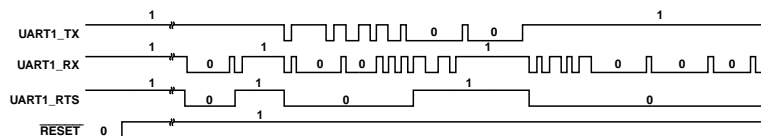


Figure 40-9: Host Relying on RTS

When the boot kernel is processing fill or Initcode blocks, it can require extra processing time and must delay the host from sending more data. This request is signaled using the RTS output.

The *Host Relying on RTS* figure shows RTS timing when an extended Initcode routine executes. Since code execution is distracting from the data loading, the host device must be prevented from sending more data. The timing of the RTS depends on the state of the `UART_CTL.RFRT` bit. This bit is cleared during UART slave boot mode when RTS is de-asserted, the UART receive FIFO contains 4 or more data words, and another start bit is detected.

NOTE: Secure Boot Stream Padding

For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

Autobaud Detection

The kernel supports autobaud detection using the '@' character as data. The host is expected to have its clock set to a rate supported in the UART.

To determine the bit rate when performing autobaud detection, use the following steps:

1. The boot kernel expects an '@' character (0x40, eight bits data, one start bit, one stop bit, no parity bit) on the UART RXD input.
2. The `UART_CLK.EDBO` and the `UART_CLK` register is cleared.
3. The boot kernel acknowledges, and the host then downloads the boot stream. The acknowledgment consists of 4 bytes: 0xBF, `UART_CLK [15:8]`, `UART_CLK [7:0]`, 0x00.
4. The host is requested to not send further bytes until it has received the complete acknowledge string.
5. Once the 0x00 byte is received, the host can send the entire boot stream.

The host knows the total byte count of the boot stream, but it is not required to know the content of the boot stream.

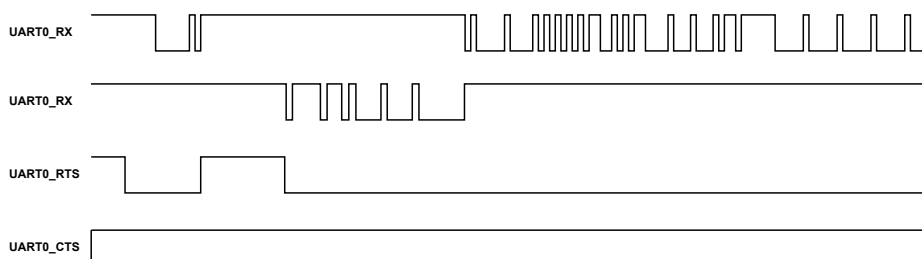


Figure 40-10: UART Autobaud Detection Waveform

The *UART Autobaud Detection Waveform* figure provides timing information for UART booting. After the bit rate is known, the UART is enabled and the kernel transmits the 4 acknowledge bytes.

NOTE: For default UART slave mode, the peripheral DMA fault, `UART2_RXDMA_ERR` with Fault ID 163 is enabled.

Run-time API

The UART slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both autobaud detection and device configuration can be disabled, and a device other than the default UART0 can be specified.

If ROM_BCMD_NOCFG flag is specified, it is the programs responsibility to configure pin multiplexing as required.

Autobaud detection can be suppressed using the ROM_BCMD_NOAUTO flag. In this case, the desired configuration can be passed through the ROM_BCMD_UART_CLK bit field. If the ROM_BCMD_UART_CLK bit field is zero, UART_CLK is evaluated. If a value of 0xFFFF was present, the default error routine of the boot kernel is called and the booting process is aborted. Otherwise, the value in UART_CLK remains untouched.

The following table provides descriptions of the adi_rom_Boot parameter.

Table 40-21: UART Slave Boot command Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16	ROM_BCMD_UART_CLK	UART Clock Divider. When set to zero this field is ignored.	
15	ROM_BCMD_UART_EDBO	UART Clock Divider Mode When set enables EDBO functionality.	
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the UART device to use.	
		0x0	UART0
		0x1	UART1
		0x2	UART2
		0x3 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, enables SPI slave boot. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.	
		0x3	UART

NOTE: All bits in the above table that are not defined must be set to zero. Supported features may be limited depending on peripheral instance.

OSPI Master Boot Mode

OSPI Master boot supports booting from flash devices via SPI3 controller. Similar to legacy SPI (SPI0/SPI1/SPI2), it supports boot from many different flash devices provided by different flash vendors further increasing the range of devices due to support for DDR modes of operation. This interface enables to reduce boot times by using faster modes of operation supported by the OSPI controller including DDR modes of operation to achieve higher transfer rates. Most flash vendors support atleast a basic and standard 0x03 read command allowing for a single bit SPI flash interface to be supported.

The OSPI boot mode also uses a device auto-detection feature that is enabled by default. This lets the boot stream itself instruct updates to the SPI configuration and the read command used allowing for more efficient transactions such as for enabling quad bus widths and DDR modes and faster SPI clocks.

Boot from External SPI Flash Devices

The OSPI boot mode supports booting from 24-bit or 32-bit addressable flash devices. The boot mode uses the MDMA channels, which works with OSPI controller to get data from flash memory in memory mapped mode..

When auto-device detection is enabled, the SPI memory is initially read using the standard 0x03 SPI read command with a reduced clocking frequency for maximum compatibility. The first nibble of the boot stream is then used to reconfigure the SPI interface.

NOTE: Support for automatic device detection via the first nibble of the boot stream is not supported when booting secure boot streams. Instead when signing the boot image an attribute can be set in the image header that specifies the configuration to use.

For booting, the SPI memory is connected as shown in the *SPI Memory Connections* figure.

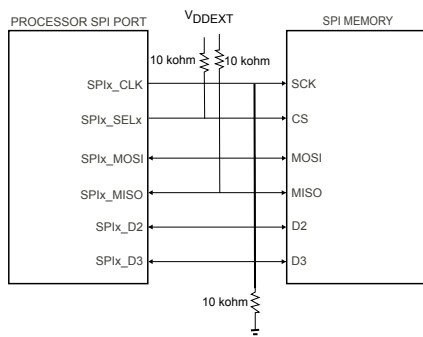


Figure 40-11: SPI Memory Connections

The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in a high-impedance mode such as during reset.

Initialization codes are allowed to manipulate the `ADI_ROM_BOOT_CONFIG::dBootCommand` to extend boot mechanism to a second SPI memory connected to another slave select pin. Updating the field that specifies the slave select signal for use, allows the boot process to manage larger boot streams than fit into a single SPI device.

NOTE: If modifying the slave select signal used during the boot process, configure the pin multiplexing to enable the correct functionality for the pin. Once the boot process has proceeded past the configuration function and the boot process has actually started, the boot kernel will not perform any further pin multiplexing operations.

OSPI Device Detection Routine

As boot mode supports booting from various SPI memories, the boot kernel automatically detects what type of memory is connected. To determine whether the SPI memory device requires a 24-bit or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The `OSPI_MISO` signal requires a pull-up resistor. The routine relies on the fact that memories do not drive their data outputs unless the right number of address bytes are received.

The SPI flash autodetection routine determines the SPI configuration to be used by data in the boot stream. The autodetection first determines the number of address cycles needed to address the flash device.

This operation is performed by configuring the controller in STIG mode initially with the number of expected address bytes programmed to 3 and a basic read command (0x3). A read command from address 0x00000000 with a 3-byte address is issued. If the flash returns a value other than 0x00 or 0xFF, the flash is in 3-byte address mode. Otherwise, the same process is repeated with the STIG controller configured with four address bytes. Only three and four byte address cycles are supported.

The lower nibble of the received byte is then used to further customize the boot mode. This nibble is referred to as the *BCODE*. The boot code applies settings to the OSPI peripheral according to the *OSPI Master Boot BCODE Descriptions*.

The boot process continues with normal boot operation and it re-issues a command to re-read from address 0. Two read sequences load the first block header. Separate read sequences load further block headers and block payload fields.

The *SPI Device Detection Principle* figure illustrates how individual devices behave.

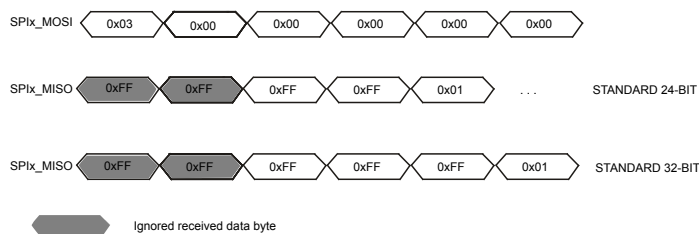


Figure 40-12: SPI Device Detection Principle

Table 40-22: OSPI Master BCODE Configuration Lookup Table

Member Name	BCODE				
	0	1	2	3	4
Transfer Type	Single bit command, address and data	Single bit command, address and data	Single bit command, address and data	Single bit command, dual address and data	Single bit command, dual address and data
ubDummyCycles	0x00	0x00	0x08	0x04	0x08
ubReadCommand	0x03	0x03	0x0B	0xBB	0xBB
ubDataBits	0x00	0x00	0x00	0x01	0x01
ubAddressBytes (This field is retained. Not used by boot kernel)	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x000F	0x000F	0x0003	0x0003	0x0003
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000
nCfg	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081
nDsr	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002
nDrir	0x00000000	0x00000000	0x00000000	0x00011000	0x00011000
uReserved1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00

Table 40-23: OSPI Master BCODE Configuration Lookup Table

Member Name	BCODE				
	5	6	7	8	9
Transfer Type	Single bit command, Quad bit address and data	Single bit command, Quad bit address and data	Single bit command, single bit DDR address and data	Single bit command, single bit DDR address and data	Single bit command, dual DTR address and data
ubDummyCycles	0x06	0x0A	0x06	0x08	0x02
ubReadCommand	0xEB	0xEB	0x0D	0x0D	0xBD
ubDataBits	0x02	0x02	0x00	0x00	0x01
ubAddressBytes (This field is retained. Not used by boot kernel)	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x0003	0x0003	0x0003	0x0003	0x0003
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000
nCfg	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081
nDsr	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002

Table 40-23: OSPI Master BCODE Configuration Lookup Table (Continued)

Member Name	BCODE				
	5	6	7	8	9
nDrir	0x00022000	0x00022000	0x00000400	0x00000400	0x00011400
uReserved1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00

Table 40-24: OSPI Master BCODE Configuration Lookup Table

Member Name	BCODE					
	A	B	C	D	E	F
Transfer Type	Single bit command, dual DTR address and data	Single bit command, dual DTR address and data	Single bit command, quad DTR address and data	Single bit command, quad DTR address and data	Single bit command, quad DTR address and data	Reserved
ubDummyCycles	0x04	0x06	0x06	0x07	0x08	0x00
ubReadCommand	0xBD	0xBD	0xED	0xED	0xED	0x0B
ubDataBits	0x01	0x01	0x02	0x02	0x02	0x00
ubAddressBytes (This field is retained. Not used by boot kernel)	0x03	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
nCfg	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081
nDsr	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002
nDrir	0x00011400	0x00011400	0x00022400	0x00022400	0x00022400	0x00000000
uReserved1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00	0x00

Table 40-25: OSPI Master BCODE Configuration Lookup Table (Applicable to Secure Streams Only)

Member Name	BCODE					
	10	11	12	13	14	15
Transfer Type	Single bit command, dual address and data	Single bit command, quad address and data	Single bit command, single DTR address and data	Single bit command, single DTR address and data	Single bit command, quad DTR address and data	Single bit command, quad DTR address and data
ubDummyCycles	0x0C	0x04	0x04	0x05	0x03	0x09

Table 40-25: OSPI Master BCODE Configuration Lookup Table (Applicable to Secure Streams Only) (Continued)

Member Name	BCODE					
	10	11	12	13	14	15
ubReadCommand	0xBB	0xEB	0x0D	0x0D	0xED	0xED
ubDataBits	0x01	0x02	0x00	0x00	0x02	0x02
ubAddressBytes (This field is retained. Not used by boot kernel)	0x03	0x03	0x03	0x03	0x03	0x03
uwClkLower	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003
uReserved0	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
nCfg	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081	0x00000081
nDsr	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002
nDrir	0x00011000	0x00022000	0x0000400	0x0000400	0x00022400	0x00022400
uReserved1	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
nDummy	0x00	0x00	0x00	0x00	0x00	0x00

NOTE: For the above configurations, the addressing scheme can be 3-bytes or 4-bytes depending on the addressing of flash detected in auto-detection. The SPI mode byte issued for all the SPI master peripheral based configurations is 0x00. The mode byte is the first byte transmitted after the address cycles and is used to control the continuous read mode functionality in which the next read operation is not required to issue a command cycle. Continuous read mode is not supported during the boot process.

Supported Quad Mode Enable Methods

The boot ROM does not support enabling quad mode on the SPI flash device. To boot in quad mode, the flash device must be configured outside the boot ROM.

To enable quad mode, initially boot in dualmode and use an initcode.

Run-Time API

The following table provides descriptions of the [adi_rom_Boot\(\)](#) command parameter.

Table 40-26: OSPI Master Boot Command Bit Descriptions

Bits	Name	Setting	Description
31:28	ROM_BCMD_SPIM_SPEED	0-1111b	SPI clock divider to be used. This value is written to the SPI peripherals clock divider register

Table 40-26: OSPI Master Boot Command Bit Descriptions (Continued)

Bits	Name	Setting	Description
27	ROM_BCMD_SPIM_CMDSK IP_EN	0	Issue a read command with every read operation
		1	Do not issue a read command with each read operation only issue the read command on the first read.
26:22	ROM_BCMD_SPIM_DUMMY	00000b	No dummy clock cycles required after the address
		00001b	1 dummy clock cycle required after the address
	
		11111b	31 dummy clock cycle required after the address
21:20	ROM_BCMD_SPIM_ADDR	00b	Flash device requires an 8-bit address
		01b	Flash device requires a 16-bit address
		10b	Flash device requires a 24-bit address
		11b	Flash device requires a 32-bit address
19:16	ROM_BCMD_SPIM_BCODE	0000b-1111b	Applies an initial configuration as described in the SPI Master Boot BCODE Configurations. Not all options may be available.
15	ROM_BCMD_SPIM_CMD	0	Issue the read command over the single bit bus
		1	Issue the read command over the multi-bit bus. Not recommended for use. Intended for use with products with on chip SPI flash only, and even then it can cause complications.
14:12	ROM_BCMD_SPIM_SSEL	000b	Use slave select 1 for SPI chip select
		001b	Use slave select 2 for SPI chip select
		010b	Use slave select 3 for SPI chip select

Table 40-26: OSPI Master Boot Command Bit Descriptions (Continued)

Bits	Name	Setting	Description
		011b	Reserved
		100b	Reserved
		101b	Reserved
		110b	Reserved
		111b	Reserved
11:8	ROM_BCMD_SPIM_DEVENUM	0 to 16	Specifies the boot peripheral enumeration. For OSPI Master boot this should be set to 3.
7	Reserved	Reserved	Reserved
6	ROM_BCMD_SPIM_NOAUTO	0	Perform automatic device detection and peripheral configuration based on the BCODE value (first nibble) of the boot streams block header
		1	Do not perform automatic device detection
5	ROM_BCMD_SPIM_NOCFG	0	Instructs the config routine to perform pinmuxing configuration and full peripheral configuration
		1	The config routine will not change the configuration of the peripheral and it will not enable pinmuxing. This can be used if the boot software does not support specific features of a peripheral, users can configure the peripheral prior to calling the boot routine.
4	ROM_BCMD_SPIM_HOST	0	Master boot mode
		1	Not to be used for OSPI Master Boot
3:0	ROM_BCMD_SPIM_DEVICE	0000b	Not to be used for OSPI boot
		0001b	Not to be used for OSPI boot
		0010b	Not to be used for OSPI boot
		0011b	Not to be used for OSPI boot
		0100b	Not to be used for OSPI boot
		0101b	Not to be used for OSPI boot

Table 40-26: OSPI Master Boot Command Bit Descriptions (Continued)

Bits	Name	Setting	Description
		0110b	Not to be used for OSPI boot
		0111b	Not to be used for OSPI boot
		1000b	OSPI Memory Mapped Boot (Memory mapped mode and MDMA)
		1001b – 1111b	Not to be used for OSPI boot

NOTE: All bits in the above table that are not defined must be set to zero.

To support the above lookup table at varying SPICLK frequency range, boot kernel programs the Read data capture register such that sampling time does not go beyond one full cycle of SPI CLK for STR and half cycle of SPI CLK for DTR mode of operation. If this timing is not sufficient to support all flash vendors, there is a provision to update the Read data capture register from Customer OTP space with worst case value corresponding to different flash device used by user. This value has to be derived based on the min-max AC timing propagation delay from OSPI controller to pin.

Boot Loader Stream

A loader stream is a set of formatted blocks containing instructions for the boot kernel, as well as the application and data for loading to the chip. This section details the different aspects of the stream, its blocks, some common use cases, and the ROM function.

Each block begins with a block header that contains attributes of the block as well as flags to control its processing by the boot ROM. On power-up or reset, the processor begins executing the on-chip boot ROM. The boot stream is either read from memory or received from a peripheral, depending on the boot mode specified. Each block in the boot stream instructs the boot kernel to perform an action, most commonly to load data to a specified location. For a complete list of actions, refer to the [Block Types](#) section.

Some common actions include the following:

- running code that initializes a peripheral
- processing data then loading it to a location

As shown in the *Project Flow* figure, a utility is required to process the resulting output from the tool chain to create a valid boot stream. This utility can be in the form of a standalone application or script that parses an application image file, elf output file, or text-based file such as Intel hex. A flash programmer utility can format a boot stream internally.

The elfloader utility parses the application data, and creates a set of blocks representing the segmented application. When processing an actual image that must be loaded to a single contiguous memory block, this representation can

contain as little as a single header block. The output of the standalone utility is stored in a loader file (.ldr). The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral such as the UART during boot time

A loader stream must always begin with a first block and end with a final block. The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral during boot time

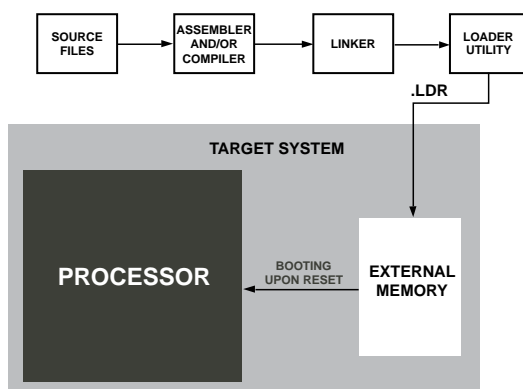


Figure 40-13: Project Flow

The *Booting Process* figure shows the parallel or serial boot stream contained in a flash memory device. In host boot scenarios, the non-volatile memory usually connects to the host processor rather than directly to the processor. After reset, the on-chip boot kernel reads and parses the headers and the loader stream is processed block-by-block. Finally, payload data is copied to destination addresses, either in on-chip L1 and L2 memory, or off-chip to SDRAM or SRAM.

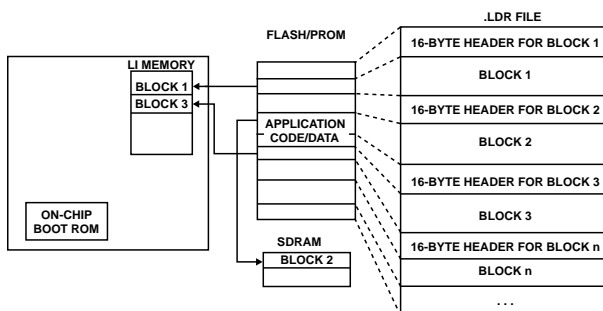


Figure 40-14: Booting Process

In some cases (for example, secure boot or when the BFLAG_INDIRECT flag for any block is set), the boot kernel uses another memory block for intermediate data storage. In order to preserve the security of the device processors do not allow these storage regions to be initialized with boot data. The boot stream is loaded to the intermediate

storage then processed by the kernel and loaded to the final destination. The final destination cannot be the intermediate storage location otherwise the boot process terminates in an error.

Block Header

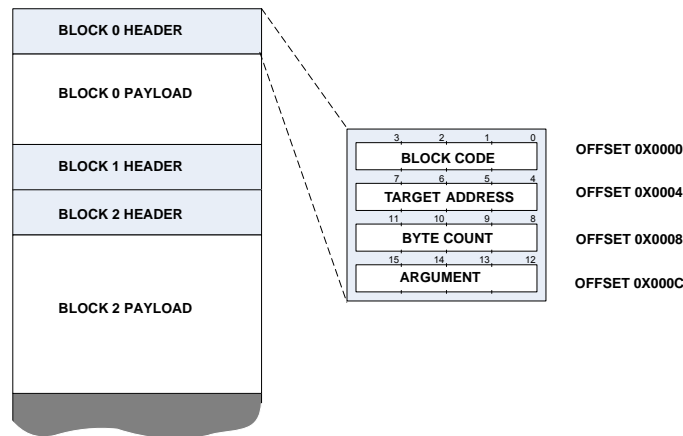


Figure 40-15: Loader Stream Block Structure

A boot stream consists of multiple boot blocks as shown in the figure. A 16-byte block header begins every block. The 16 bytes are functionally grouped into four 32-bit words:

- Block code
- Target address
- Byte count
- Argument field

This section provides a general description describes the fields in general. The use varies depending on the block type and boot mode.

Block Code

Table 40-27: Block Code Flags

Bit	Name	Description
0–3	BICODE	Specific to boot modes (see Boot Modes)
4	BFLAG_SAVE	Intended to allow for a user application to mark blocks for saving the memory of this block to off-chip memory in case of power failure. The on-chip boot kernel does not use this flag.
5	BFLAG_AUX	When set indicates that the byte address space translation for SHARC core boot blocks requires translation to the 48-bit PM address space. When cleared translation to the 32-bit address space is performed.
7–6	Reserved	
8	BFLAG_FILL	Fill the target location with a specified 32-bit value

Table 40-27: Block Code Flags (Continued)

Bit	Name	Description
9	Reserved	
10	BFLAG_CALLBACK	Calls the previously registered callback function
11	BFLAG_INIT	Calls function at target address. If the block contains a payload, the payload is loaded prior to the call.
12	BFLAG_IGNORE	Block payload is ignored
13	BFLAG_INDIRECT	Boots the payload to the intermediate storage location
14	BFLAG_FIRST	Indicates the block that is the beginning of a new application
15	BFLAG_FINAL	Indicates the last block of a loader stream. Booting will complete after processing the block. This flag does not denote the end of an application in a Multi-Application Boot Stream.
16–23	HDRCHK	A simple 8-bit XOR checksum of the other 24 bits in the boot block header.
24–31	HDRSIGN	0xAD, 0xAC or 0xAB. Indicates if the boot block is intended for core 0, core 1 or core 2 respectively.

TARGET_ADDRESS

The `TARGET_ADDRESS` holds the applicable address for the block, (where the code or data is loaded). However, the interpretation of the field differs depending on what specific flags are set in the block code. Refer to the documentation for each block type for details.

The following attributes must be true:

- The target address must be divisible by 4, as the boot kernel uses 32-bit DMA for certain operations.
- The target address must point to valid on-chip or off-chip memory locations.

BYTE_COUNT

The `BYTE_COUNT` must be divisible by 4, and can also be zero. This 32-bit field generally holds the size of the block. In some cases, it has a different use (such as when `BFLAG_FILL` is set). See the [Block Types](#) section for information on the variations.

ARGUMENT

The 32-bit field is a user-variable for most block types. The `Initcode` or `callback` routine can access this value and use it for optional instructions.

The different block types use the `ARGUMENT` field in various ways. See the [Block Types](#) descriptions for further information.

Block Types

A loader stream is a set of linked blocks and each block is responsible for performing a certain function dependent on the block type. The flags in the block header define a block type. Operations include functions such as loading data, filling a memory region with data, and instructing the kernel to stop processing. This section describes each block type and its usage within a boot loader stream.

Normal Block

The primary function of a block is to load data into a specified location of memory. A normal block instructs the boot kernel to load the data contained in its payload to the location specified in the `TARGET_ADDRESS` field. The `BYTE_COUNT` defines the size of the payload. Once the correct amount of data has been loaded, the kernel moves on to process the next block in the stream.

Table 40-28: Flags

Flag	Required Value	Init
<code>TARGET_ADDRESS</code>	Y	Address where payload is loaded (must be valid)
<code>BYTE_COUNT</code>	Y	Size of block in bytes

First Block

A first block indicates the start of a boot stream and is always needed at the beginning of the boot stream. When a loader stream contains [Multi-Application Boot Streams](#), a first block that occurs within the loader stream indicates the beginning of a new application.

When the kernel processes the first block in a loader stream, the boot kernel uses the `TARGET_ADDRESS` to determine the application entry location. For more details, refer to [Boot Termination and Application Execution](#).

NOTE: A first block cannot be combined with a fill block.

Table 40-29: Flags

Flag	Required Value	Init
<code>BFLAG_FIRST</code>	Y	1
<code>ARGUMENT</code>	Y	Offset to the next application, or first address following loader stream. Commonly referred to as the <code>NEXTDXE</code> field.
<code>TARGET_ADDRESS</code>	Y	When the block is the first block in a loader stream it defines the start address for the application. If the block is not the first in a loader stream, use the target address as in normal operation.

Final Block

The final block marks the last block in a boot stream. After processing a final block, the boot kernel jumps to the start address of the application. For more information on the definition of the start address, refer to [Boot Termination and Application Execution](#).

There is further customization available to alter the kernel behavior. For example, the kernel can be instructed to return from the boot routine rather than jump to the application using initialization codes or the `adi_rom_Boot()` API.

Before the boot kernel passes program control to the application, it does some housekeeping. Most of the registers in use are set to their default state. However, some register values can differ depending on the boot mode. See [Boot Modes](#) for more information.

Table 40-30: Flags

Flag	Required Value	Init
BFLAG_FINAL	Y	1

Indirect Block

An indirect block is first loaded to a storage location before being copied to the destination. This block is used in the following situations:

- Some boot modes do not use DMA from the boot peripheral. The core is not always able to access some memory locations directly or efficiently. An intermediate load to a different location improves overall efficiency.
- In some booting situations, the data in the payload must be operated-on or analyzed before it is fully loaded (such as decryption or a checksum calculation). By using an intermediate location, such situations are simplified and are more efficient when loading to off-chip memories.

In some cases, a boot block does not fit into temporary storage memory. Having a larger buffer can improve boot performance. If an entire block cannot fit into the buffer, it is processed in pieces. Initialization code or callback functions can alter the temporary buffer region, including its location and size, by modifying the `ADI_ROM_BOOT_CONFIG::pTempBuffer` and `ADI_ROM_BOOT_CONFIG::dTempByteCount` variables in the [struct ADI_ROM_BOOT_CONFIG](#) structure.

Table 40-31: Flags

Flag	Required Value	Init
BFLAG_INDIRECT	Y	1
BFLAG_CALLBACK	N	Defines a callback function to operate on intermediate data. These 2 flags are often used together.

Ignore Block

It is a block that is ignored by the loader stream (in several cases). These blocks are useful when it is not possible to pass information in another block header. For example, if the first block contains data such as a firmware revision rather than application code, then `BFLAG_IGNORE` can be set with the correct application start address. This step ensures that the loader stream uses the correct start address. Since this block has no other function, identify it as an ignore block. Then, the kernel does not attempt to process any payload.

Ignore blocks clear the following flags, disabling the corresponding blocks from being processed if set along with `BFLAG_IGNORE`:

- `BFLAG_INIT`
- `BFLAG_CALLBACK`
- `BFLAG_FINAL`
- `BFLAG_AUX`

NOTE: When `BFLAG_IGNORE` is set along with `BFLAG_FIRST`, only the payload associated with the first block is ignored. The application entry point retrieved from the first block is always processed.

Table 40-32: Flags

Flag	Required Value	Init
<code>BFLAG_IGNORE</code>	Y	1
<code>BYTE_COUNT</code>	Y	Size of block to ignore; can be zero

Fill Block

A fill block instructs the boot kernel to perform a 32-bit memory fill of the memory region. Fill blocks help minimize the size of a boot stream when an application contains large arrays of data that need are initialized at startup. Zero fill is the most common fill block type. However, any 32-bit fill value is supported.

Table 40-33: Flags

Flag	Required Value	Init
<code>BFLAG_FILL</code>	Y	1
<code>TARGET_ADDRESS</code>	Y	Address where payload is loaded (must be valid)
<code>BYTE_COUNT</code>	Y	Size of block in bytes (must be multiple of 4)
<code>ARGUMENT</code>	Y	32-bit fill value

Init Block

An initialization block instructs the boot kernel to perform a function call to the target address after the entire block has loaded. The function called is referred to as the *initialization code (Initcode) routine*. Refer to the API reference [initcode\(\)](#) for full details.

If the Initcode routine has been previously loaded, the block can declare a zero-size and have no payload.

Initcode routines can be used to speed up and customize booting mechanisms exposed by the boot kernel. Traditionally, an Initcode routine is used to set up the system PLL, bit rates, wait states, and the external memory controllers. Boot time can be significantly reduced when an init block is executed early in the boot process.

Initcode routines are required to follow the C language calling conventions. The expected C prototype is:

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

NOTE: When programming in assembly, use a return from subroutine instruction for returns.

The structure provided to the Initcode routine by the boot kernel contains information about the block being processed. It includes header information, locations of temporary block data (for indirect blocks), target address, and byte count. See [struct ADI_ROM_BOOT_CONFIG](#) for a full list and details on the provided data.

In the simplest case, an Initcode routine consists of only a single block in which the `BFLAG_INIT` flag is set. For larger routines, a sequence of blocks can incrementally load the routine, and only the last block sets the `BFLAG_INIT` flag. In the latter case, the last block has no payload attached, and simply instructs the boot kernel to issue a call to subroutine instruction.

An Initcode routine can be overwritten by a successive block when it is no longer needed. Otherwise, the routine can be called at multiple points during the boot process, and even remain in memory after the application completes booting.

NOTE: The following list provides requirements for Initcode that is written in C or C++.

- Ensure that the Initcode routine does not contain calls to the run-time libraries
- Do not assume that parts of the run-time environment, such as the heap, are fully functional
- Ensure that all run-time components are loaded and initialized before the routine executes

The loader utility and tool set provide mechanisms to aid in implementing initialization codes and organizing them properly within the boot loader stream. A special project type is provided to allow the creation of Initcode routines as separate projects. Options are available to assign particular pieces of the application to be Initcode routines. For details and more information on the utility, see the *Loader and Utilities Manual*.

Table 40-34: Flags

Flag	Required Value	Init
BFLAG_INIT	Y	1

Table 40-34: Flags (Continued)

Flag	Required Value	Init
TARGET_ADDRESS	Y	Location to load payload data. Call to subroutine issued to the same location.
ARGUMENT	N	Can be used to supply block specific arguments
BYTE_COUNT	Y	Size of payload; can be zero

NOTE: Init blocks cause software not located in the boot ROM during the boot process to execute. In the case of a secure boot scenario, initcode routines are not supported. The secure boot authentication process is performed at the end of the boot process. Execution of any user software prior to the authentication process violates the secure boot requirements.

Callback Block

It instructs the boot kernel to call a pre-registered function when the payload of the block is loaded. The purpose of a callback routine is to apply standard processing to the block payload. The callback routine is registered through an Initcode routine prior to loading a block using the routine. Typically, callback routines contain checksum, decryption, decompression or hash algorithms.

To register a callback, create an [Init Block](#) whose Initcode modifies the `ADI_ROM_BOOT_CONFIG::pCallbackFunction` with the address of the callback function to execute. A callback function must be registered prior to processing a callback block.

As callback routines require access to the payload data of the boot blocks, load the block data before processing. Often an [Indirect Block](#) is used in combination with a callback block. Indirect blocks in combination with callback blocks allow for post processing of the loaded data before it is then transferred to the final destination.

Callback routines are expected to meet the C language calling conventions. The prototype is as follows:

```
ROM_BOOT_RESULT callback(
    ADI_ROM_BOOT_CONFIG * pBootConfig,
    ADI_ROM_BOOT_BUFFER * pBuffer,
    uint32_t nFlags
)
```

The `pBootConfig` argument contains a pointer to the [struct ADI_ROM_BOOT_CONFIG](#) information and `pBuffer` provides access to the address and size of the block (can vary when using indirect). The `nFlags` parameter is specifically used when the `BFLAG_INDIRECT` flag is also used. The `CBFLAG_DIRECT` flag indicates that the `BFLAG_INDIRECT` bit is not active so that the program only calls the callback routine once per block. When the `CBFLAG_DIRECT` flag is set, the `CBFLAG_FIRST` and `CBFLAG_FINAL` flags are also set.

NOTE: Callback blocks result in execution of software that is not located in the boot ROM during the boot process. In the case of a secure boot scenario callback routines are not supported because the secure boot authentication process is performed at the end of the boot process and execution of any program prior to the authentication process violates the secure boot requirements.

Callback Block in Conjunction with Indirect Block

A block using a callback routine is also loaded indirectly and there are slight behavioural differences. The procedure for loading is:

1. Load data into the temporary buffer defined by `ADI_ROM_BOOT_CONFIG::pTempBuffer`.
2. Issue a call to `ADI_ROM_BOOT_CONFIG::pCallbackFunction`.
3. After the callback routine returns and if the return value is zero, the memory DMA copies data to the destination.

When a block does not fit entirely into the temporary buffer, loading is performed similar to indirect blocks. The software calls the callback function after each chunk is loaded into the temporary storage. The `nFlags` parameter provides information on the specific iteration.

When a block does not fit entirely into the temporary storage area, `nFlags` tells the callback routine whether it is invoked for the first time (`CBFLAG_FIRST`) or called the last time (`CBFLAG_FINAL`) for a specific block.

When the software invokes DMA to copy the data, it relies on the supplied `pBuffer` data, not the `ADI_ROM_BOOT_CONFIG::pTempBuffer` and `ADI_ROM_BOOT_CONFIG::dTempByteCount` members of the boot structure.

The callback routine can control the source of the memory DMA by altering the content of the `pBuffer` structure. This alteration is necessary when the callback routine performs data manipulation such as decompression.

When the software uses an indirect block, the return value of the callback routine determines whether the DMA transfer occurs. If the value is non-zero, then the transfer does not occur.

Table 40-35: Flags

Flag	Required Value	Init
<code>BFLAG_CALLBACK</code>	Y	1

NOTE: Callback blocks cause software not located in the boot ROM during the boot process to execute. In the case of a secure boot scenario callback routines are not supported because the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process violates the secure boot requirements.

Save Block

A save block is intended to mark blocks in a boot stream for saving to off-chip memory. The on-chip boot kernel does not use this flag. A program can process the boot stream to find address of regions of memory for saving to

external memory. On a reboot the program may then restore the previously saved contents. This block provides a means of doing a context restore after a reboot.

Table 40-36: Flags

Flag	Required Value	Init
BFLAG_SAVE	Y	1

Single-Block Boot Streams

The simplest boot stream consists of a single block header and one contiguous block of instructions and optionally data. When the appropriate flags are set in the block header, the kernel loads the block to the target address, terminates the boot process, and begins executing from the entry address of the application.

The *Initial Header for Single-Block Stream* table shows an example of a single-block boot stream header with settings that can be loaded using any boot mode. The BFLAG_FIRST and BFLAG_FINAL flags are both set at the same time. The desired location and size of the application determines the target address and byte count.

Table 40-37: Initial Header for Single-Block Stream

Field	Description of Value
BLOCK_CODE	0xAD000000 XORSUM BFLAG_FINAL BFLAG_FIRST
TARGET_ADDRESS	Start address of block
BYTE_COUNT	Number of bytes in the block
ARGUMENT	Functions as next-application pointer in multi-application boot streams.

Direct Code Execution

Applications can avoid long boot times and execute code from flash or SDRAM memory with minimal processing by the boot kernel. This feature is called direct code execution.

An initial boot block header is required for the processor to fetch and execute program code from the boot device as early as possible. The block uses safety mechanisms to avoid unpredictable processor behavior when boot memory is not yet programmed with valid data. Safety mechanisms include the header signature and the byte-wise XOR checksum. Rather than blindly executing code, the boot kernel first executes the pre-boot routine for system customization. It then loads the first block header and checks it for consistency. If the block header is corrupt, the boot kernel calls the error handler and does not start code execution.

If the initial block header check is good, the boot kernel interrogates the block flags. If the BFLAG_FINAL flag is set, the boot kernel terminates and executes the sequence as described in the [Boot Termination and Application Execution](#) section. If the application requires that the boot kernel customize the starting address in advance, the first block must also have the BFLAG_FIRST flag set. The target address field is then saved as the application start address.

For example, when the block header described in the *Direct Code Execution Header* table is placed at address 0x20000000, the boot kernel is instructed to issue a JUMP command to address 0x20000020.

Table 40-38: Example Direct Code Execution Header

Field	Value	Comments
BLOCK_CODE	0xAD7BD006	0xAD000000 XORSUM BFLAG_FINAL BFLAG_FIRST BFLAG_IGNORE BCODE
TARGET_ADDRESS	0x20000020	Start address of application code. Provided as an example this would be dependent upon the start address required for a given product.
BYTE_COUNT	0x00000010	Ignores 16 bytes to provide space for control data such as version code and build data. This field is optional and can be zero. The payload is ignored due to the BFLAG_IGNORE flag being set.
ARGUMENT	0x00000010	Functions as next-application pointer in multi-application boot streams.

Multi-Application Boot Streams

This section describes loader streams that contain multiple applications.

A boot stream is typically generated from an application file. Loader streams with more than one application are commonly referred to as multi-application booting. A loader utility often accepts multiple application files as input parameters and generates a contiguous boot image. The subsequent applications are appended to the first.

The loader utility must also update the argument field of all BFLAG_FIRST blocks. The argument field of a BFLAG_FINAL block is called the *next-application* pointer.

The next-application pointer of the first application boot stream is a relative pointer to the start address of the second application boot stream. A multi-application boot image can be seen as a linked list of boot streams. The next-application pointer of the last application boot stream is a relative pointer to the next free address. The *Multi-Application Boot Stream Example* figure shows an example.

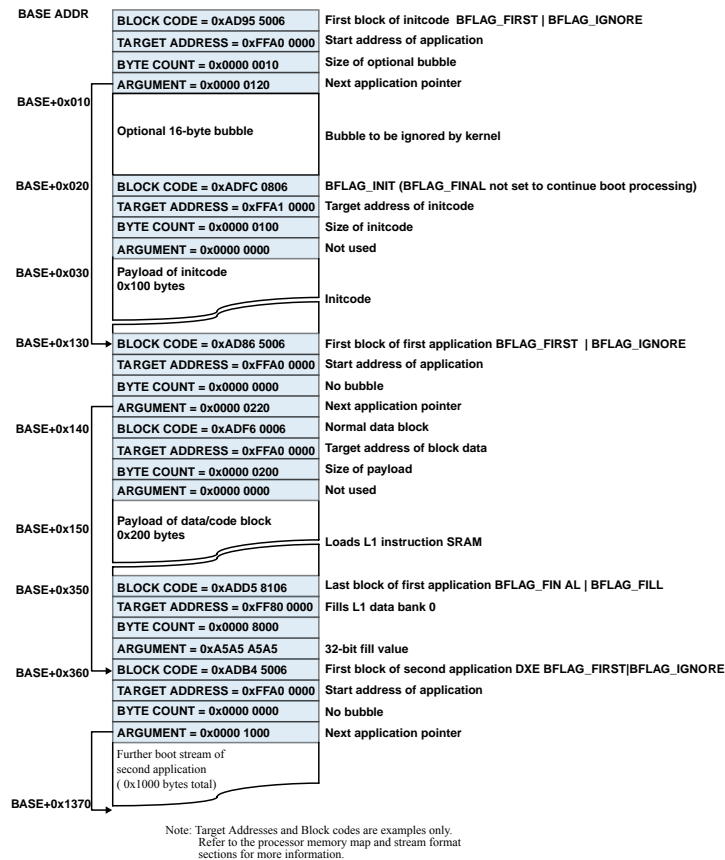


Figure 40-16: Multi-Application Boot Stream Example

The *Multi-Application Direct Code Execution Example* figure shows a linked list of initial block headers. The blocks instruct the boot kernel to terminate immediately and to start code execution at the address provided by the target address field of the individual blocks. There is nothing in the boot code that prevents multi-application streams from mixing regular boot streams and direct code execution blocks.

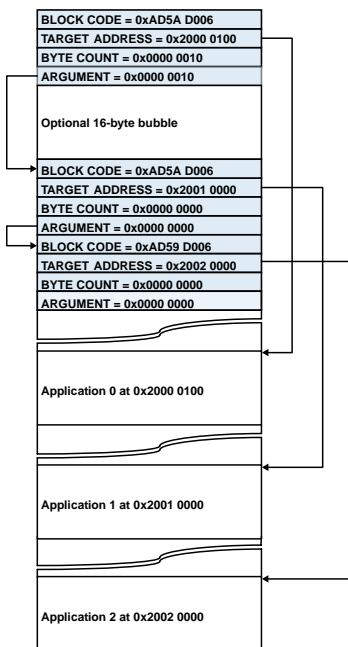


Figure 40-17: Multi-Application Direct Code Execution Example

CRC32 Protection

The boot kernel has mechanisms that allow verification of each block's payload using a 32-bit CRC. The boot ROM contains a function that can be called as an initcode to register the CRC callback and initialize the CRC peripheral with a specified polynomial. CRC32 protected loader streams are supported by the programming tools utilities.

To enable this feature an [Block Types](#) must be located in the boot stream with a TARGET_ADDRESS that points to the [adi_rom_Crc32Init\(\)](#) function in the ROM. The ARGUMENT field contains the CRC32 checksum polynomial that is used to initialize the CRC lookup table. Once the CRC initcode function in the ROM has been executed, CRC verification is enabled for all subsequent blocks except:

- Ignore
- First

NOTE: CRC functionality is dependent upon the use of an Initblock and is not supported in secure booting.

Secure Boot

The secure boot process provides a way to integrate security in the processor boot sequence. A chain of trust is established within the system by ensuring the integrity and authenticity of the boot image. Confidentiality is also supported.

Secure boot increases protection against malicious, unsecured accesses to critical and confidential resources of the processor. The boot stream application code and data must be digitally signed in order to build up a chain of trust in

the system. This requirement allows the processor to distinguish between authentic and trusted code from non-authentic and untrusted code.

Secure boot also provides confidentiality support. The digitally signed boot image can be optionally encrypted. When loading an encrypted image, the ROM decrypts while loading, then authenticates, before any application code is executed.

Secure boot is an optional feature of the processor that is disabled by default. The feature is enabled using the OTP lock API; secure boot cannot be disabled after it has been enabled. When security is enabled, developers are not dependent on Analog Devices to provision the devices, sign code or provide security certificates. The required tools for signing and encrypting the boot images are provided with the processor development tools.

Integrity and Authenticity Protection

Integrity protection is based on the secure hash SHA-2 224/256-bit algorithm. Authenticity protection is based on the ECDSA algorithm.

ECDSA uses public key cryptography consisting of two keys: a private key and a public key. The public key is stored in OTP memory on the processor so that the secure boot process can verify the authenticity of the signed boot image. Only parties in possession of the private key are able to sign the images.

Confidentiality Protection

Confidentiality protection uses the AES algorithm. Two variants are supported: wrapped and unwrapped.

The wrapped variant uses a 128-bit Key Encryption Key (KEK) stored on the processor to decrypt the 128-bit AES decryption key embedded in the secure header. The unwrapped variant stores the AES description key on the processor and utilizes it to decrypt the entire image.

The privacy of the key stored on the device (whether AES or KEK) is paramount to the security of the system. Disclosure of this key compromises security of the entire system.

Anti-Cloning Protection

Anti-cloning protection is based on the confidentiality protection. If each processor in a system uses a unique private key for confidentiality protection, then cloning between these devices can be prevented. The boot image is incompatible with devices using a different private key for decryption.

Anti-Rollback Protection

The secure boot process supports anti-rollback protection using a 32-bit counter in OTP memory. A value of 0×00000000 in the OTP makes anti-rollback disabled by default. If anti-rollback protection is required, then the program can set the rollback ID when signing the boot image. When the boot image is authenticated, the secure boot software updates the counter in OTP (if the rollback ID in the boot image is greater than the value currently stored in the OTP counter).

The rollback ID stored in the secure boot image header is integrity-protected and cannot be altered.

NOTE: To enable anti-rollback protection for secure boot operations, write a non-zero value to the 32-bit counter in OTP memory. As long as the counter register remains at the default value of zero, anti-rollback protections are not enabled regardless of the rollback ID located in a secure boot stream.

CAUTION: There are a number of restrictions for the rollback ID in the OTP module. Programs should only use the OTP boot program ROM API to set the counter. Refer to the OTP counters section for information on the implementation strategy.

Terminology

ECDSA

Elliptical Curve Digital Signature Algorithm

BLp

Boot Loader plain text, Plaintext Format

BLx

Boot Loader without key, Keyless Format

BLw

Boot Loader wrapped, Wrapped Format

BLe

Boot Loader encrypted, Encrypted Format

SBLS

Secure Boot Loader Stream

SBH

Secure Boot Header

SBCR

Secure Boot Confidentiality Root

AES

Advanced Encryption Standard

Signing for Secure Boot Images

All boot images must be digitally signed to create secure boot images. The boot image is processed by the security utilities included with the development tools to sign and optionally encrypt the boot image. The security utilities operate with key-pairs consisting of a private and a public key. The private key is used for signing the images, and the public key is used to validate an image being loaded into the processor.

CAUTION: The private key generated from the signing utility, used for signing images, is never required by the processor for successful secure boot. The private key is only ever required by the signing utility and should be made available only within the system responsible for the image signing process.

The image signing utility provides the following functionality:

- Signing and encrypting of images
- Generation of ECDSA key pairs
- Generation of random encryption keys
- Extraction of the public key from an ECDSA key pair
- Setting [Secure Boot Image Attributes](#)

For more information on the use of the signing utility, refer to the Loader and Utilities manual.

Secure Boot Image Types

This section provides an overview of the different image types and how to use them.

Plaintext (BLp) Format

The BLp format provides integrity plus authentication protection of the boot image. The boot image is produced using 224-bit or 256-bit Elliptical Curve Digital Signature Algorithm (ECDSA) private key. To authenticate the image, program the corresponding public key into the OTP `public_key` field using the OTP boot program API.

SBH	Boot Loader Stream
-----	--------------------

Wrapped (BLw) Format

The BLw format provides the highest level of protection: integrity plus authentication, confidentiality, and anti-cloning protection. The image contains a wrapped image encryption key (denoted by [K]) within the secure header. The image data is encrypted with the wrapped key, preventing cloning. An extra key is required to unwrap the header; program this key into the OTP `pvt_128key` field using the OTP boot program API.

SBH [K]	Encrypted Boot Loader Stream
---------	------------------------------

Keyless (BLx) Format

The BLx format is similar to the BLw format except that the image does not contain the key at all. This format provides anti-cloning protection only if the secure key is unique per device. Program the decryption key for the data into OTP `pvt_128key` field using the OTP boot program API.

SBH	Encrypted Boot Loader Stream
-----	------------------------------

Secure Boot Image Format

Secure Boot images provide authenticity and integrity protection during the boot process. A secure boot image is comprised of a secure boot header and an optionally encrypted loader stream.

Signed images consist of the following sections to comprise a complete secure boot image:

- Secure Boot Header
- Image Attributes
- Image Section

The [Figure 40-18 Secure Boot Image](#) shows that the image attributes are encapsulated within the secure boot header. The image attributes are actually integrity protected along with the image section. The image section contains a standard boot loader stream. Some block types are not allowed as described in [Unsupported Boot Stream Blocks](#).

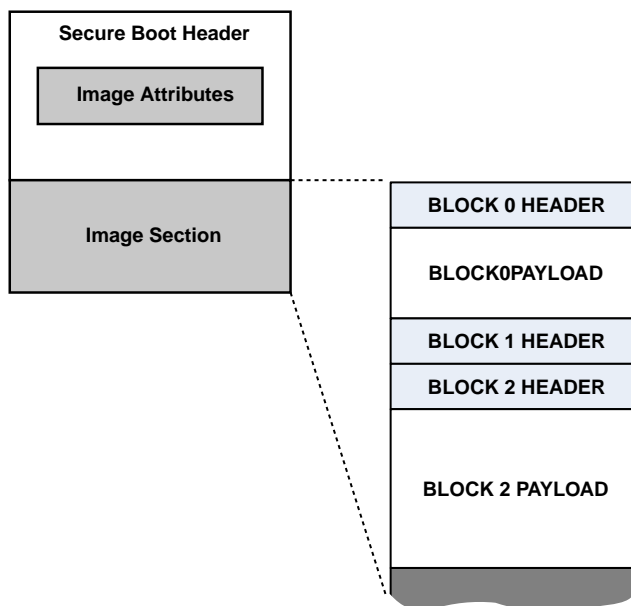


Figure 40-18: Secure Boot Image

Secure Boot Header

Table 40-39: Secure Boot Header

Bytes	Name	Description	Values		
			Keyless Format BLx	Wrapped Format BLw	Plain Text Format BLp
3:0	Type	Format and version of the image. Upper 24 bits is the image format and lower 8 bits is the image version	0x424C7801	0x424C7701	0x424C7001
67:4	Signature	The ECDSA signature of the image	Two 256-bit numbers		
91:68	Key	Confidentiality (only applicable for certain formats)	Reserved	192-bit AES-WRAP data holding a 128-bit AES key	Reserved
107:92	IV	Initialization Vector (only applicable for certain formats)	16-bit IV generated during signing process		Reserved
111:108	Length	The length of the image section in bytes	Maximum supported byte count 0x10000000 bytes		
175:112	Attributes	Image attributes	Support for up to 8 image attributes		
179:176	Reserved				

Secure Boot Processing Overview

The *Secure Boot Processing* figure illustrates how the block is processed using secure features (not all block header type details are shown). For details on the various block types and their function, see [Block Types](#). Some image types are decrypted. Decrypt indicates that the data is decrypted when applicable to the [Secure Boot Image Types](#) at that particular stage.

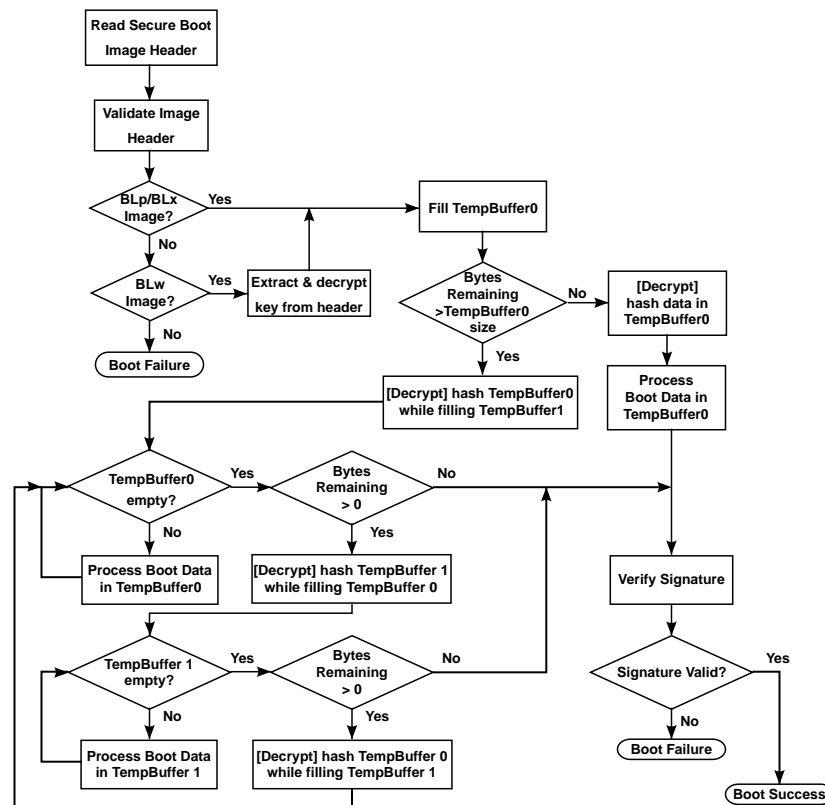


Figure 40-19: Secure Boot Processing

Unsupported Boot Stream Blocks

To ensure the security of the processor, the following block types are not supported in a secure boot image. If the boot kernel finds one of these block types, the boot process terminates.

- **Init Block:** It requires a call to the application code prior to the authentication of the boot image. If customizations or optimizations are necessary to improve the load performance, use a second stage loader style implementation. The first application contains only the custom code. Issue a call using the `api_boot()` routine to boot using the desired device.
- **Call Block:** It requires a call to a user-defined address prior to the authentication of the boot image, and therefore cannot be supported.

NOTE: Secure boot streams use double buffer [Page Mode](#) to optimize the boot process. This functionality allows for the performance of decrypt and hash operations on received data while new data is fetched from the boot source. This host in slave boot mode must ensure that more data is sent after the boot stream to ensure that the temp buffer is filled completely. The size of the secure boot stream minus the size of the secure boot header must be a multiple of the size of the temp buffer. The temp buffer default size is 1024 bytes.

Secure Boot Image Attributes

Secure boot image attributes form part of the secure boot header. The attributes provide more information about the content of the secure boot image.

All image attributes are integrity protected using the same algorithm as the image section. When the image authentication process completes and the image is successfully authenticated, the image attributes are known to be trustworthy.

Attributes are specified as type value pairs with both the type and value being a 32-bit value. The boot code supports the following image attributes.

Table 40-40: Secure Boot Image Attributes

ID	Name	Description	Values	
0x00000000	Unused	Unused attribute	0x00000000	
0x00000001	Version	Version of the attribute format	0x00000000	
0x00000002	Rollback ID	Current value of the rollback counter	0x00000000	Rollback disabled
			0x00000001 - 0x0000001F	Current firmware revision. It must be greater than or equal to the value retrieved from OTP.
0x80000001	NoRestore	Controls whether the boot process restored registers back to default values and clears sensitive security related information from the stack and dedicated structures.	0x1	Do not restore registers and clear security information from stack and dedicated structures.
			Other	Restore registers and clear security sensitive information from the stack and dedicated structures.
0x80000002	BCODE	Used by the boot mode drivers that support auto-detection to configure the device from a range of preconfigured settings	Values between 0x00000000 and 0x0000000F supported.	
0x80000003	ECDSA Type	Used by boot ROM to perform Authentication using ECDSA-224 or ECDSA-256 algorithm.	224 : Authentication using ECDSA-224 algorithm 256 : Authentication using ECDSA-256 algorithm	

Secure Boot Streams

The secure boot image sections contain the code and data for loading to the various memory regions of the processor. The content is in the form of a boot stream format consisting of block headers that provide a descriptor for the associated block payloads.

The image section within the secure boot stream consists of a standard [Boot Loader Stream](#) consisting of block headers and payloads as generated by the supporting tools and utilities.

The secure boot image section can contain application images for just a single core or for multiple cores allowing for a flexible booting strategy.

A single boot image containing program and data for multiple cores permits a full system boot initializing all internal memories without needing a second stage loader approach. If boot images are required to initialize external memories, then a multi-stage loader approach can be required to configure the memory interfaces. The advantage of a single boot image for multiple cores is that authentication and decryption of the boot image requires only one execution. However, it is over a larger boot image.

Multiple single-core boot images permit one core to boot then execute that core's boot image without booting the other cores. This booting strategy can be needed when a single processor must be brought up quickly to deal with some initial tasks before booting the rest of the system. Or, it can be used to initialize extra peripherals to use for external memory interfaces. The core that has previously booted can then control the boot process for the remaining cores.

Single Core Images

Single core boot images are the result of processing a single core's output from the linker and converting it to a compliant boot stream. The resulting boot stream has a single first block at the beginning of the boot stream and a final block at the end of the boot stream.

The [Block Header](#) HDRSIGN field of the resulting boot stream is used to identify which core the image is intended for. This identification is required so that the boot code can set the correct `RCU_SVECT0` from the target address in the first block once the authentication is successful. When the boot image is loaded and authentication is successful, the boot code jumps to the location stored in the cores corresponding `RCU_SVECT0` register.

This boot stream must then be signed using the secure boot utilities resulting in the final secure boot image for a single core.

A second stage loader option is also available giving maximum flexibility both in terms of using ROM functionality, or creating a custom booting strategy. The simplest option is to have the application use `adi_rom_Boot()` to boot the main application image or indeed a third stage loader when one is required.

The boot stream is generated such that all image contents are in the system address space. Core specific L1 memory sections are converted during the boot stream generation process such that they are loaded through the multiprocessor memory space. The core executing the second stage loader can boot an application image intended for another core.

NOTE: The `ROM_BFLAG_RETURN` flag is set when calling the boot routine for the other cores than the core executing the second stage loader then cleared when loading the cores own image. Failure to do so would result in unintentional behavior.

When the core is used to boot a separately generated boot stream for another core, the `ROM_BFLAG_RETURN` flag should be used when calling `adi_rom_Boot()` to instruct the boot kernel to return to the calling application,

which in this case is back to the core that was running the second stage loader. Failure to set this flag results in the core vectoring to the location in its own `RCU_SVECT0` register which is not be the intention.

By default when implementing a scheme such as previously described where one processor is responsible for booting images for another core, the other cores in the system will remain in their existing idle state that they should be placed in prior to the boot commencing. To allow the cores to execute new application, the core that is responsible for booting the other cores must reset the cores via the `RCU_CRCTL` then release them again from reset. If there is a requirement to release the core from the reset state, then this must be done within the application code of the running core.

Releasing other cores from reset to run application software while other processors are running a boot process requires careful system design. The drivers used by the boot kernel for the boot process assume the various peripheral and infrastructure resources such as MDMA channels and peripherals are available for use, and are not being used by another core. If the boot routine is being executed while other cores are running applications, then those applications must ensure that all the required boot resources are freed up and remain free in order for the boot process to complete on the remaining cores.

Secure Debug Access

The TAPC controller provides a way to restrict access to secure resources of the processor. Secure access through the debug port is protected through a 128-bit security key that must match a key that has been loaded into OTP for access.

To access a locked processor, the TAPC must allow access to the part. The TAPC only allows access to the part if it is provided with a matching key to the data loaded into its `TAPC_USERKEYn` registers.

With the processor in a locked state, on initial boot the boot ROM reads the 128-bit `secure_emu_key` from the OTP memory and programs the key into the `TAPC_SDBGKEY0` through the `TAPC_SDBGKEY3` registers before then setting the `TAPC_USERKEY_CTL.USERKEY_VALID` bit. The TAPC is then able to access a matching outside key to allow access.

CAUTION: A 16 bit non-zero value programmed on the `emu_key_disable` field in OTP OR a key of `0xFFFFFFFF`, `0xFFFFFFFF F`, `0xFFFFFFFF F`, `0xFFFFFFFF F` programmed in secure debug key field in OTP results in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by the program. If the processor fails to boot (perhaps due to corrupted firmware), there is no debug access. The only way to gain access is to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger

The key is set in OTP using the OTP boot program API to program the `secure_emu_key`. This key is read and loaded by the ROM in the following sequence:

1. Bits 31:0 of the key in OTP are stored to `TAPC_SDBGKEY0` bits 31:0
2. Bits 63:32 of the key in OTP are stored to `TAPC_SDBGKEY1` bits 31:0
3. Bits 95:64 of the key in OTP are stored to `TAPC_SDBGKEY2` bits 31:0

4. Bits 127:96 of the key in OTP are stored to `TAPC_SDBGKEY3` bits 31:0

Once the ROM has loaded the user key, a test key can be provided to the TAPC through JTAG. Refer to the Emulator manual for details for providing the key.

A key failure indication can be detected through the `TAPC_SDBGKEY_STAT` register. The boot code does not check the key status, nor does it enable any associated interrupts to signal key failure. The boot code continues to boot upon a key failure in a secure manner. The key failure status remains intact so that the application loaded can check for a failed challenge on the debug port.

The boot code can be configured to bypass the loading of the key during the boot sequence by setting the value of the `secure_emu_key` in the OTP to all ones (see `struct otp_data`). In this case, the only way to gain access to the secure resources through the debug port is to load an alternate key using the application. The alternative key must always reside in a secure region of memory. Or, if sent remotely, it should be transmitted over a secure connection.

Boot ROM Errors and Failures

Any errors encountered during the boot process results in the boot ROM jumping to the [Error Handler](#) with a specific error code written to the `RCU_MSG` register. This includes decryption failures, authentication failures, and configuration errors. The error can be a boot specific error or product specific error. The *Boot ROM Exceptions* table shows the different types of errors supported by the boot ROM.

WARNING: In secured booting, an incorrect secure debug key supplied during boot does not cause a boot failure and the processor continues to boot as normal. A program that supplies an incorrect key is unable to gain access to any secure resources of the processor.

Table 40-41: Boot ROM Exceptions

Error Name	Error ID	Description
ROM_BOOT_ERR_GENERAL	0x0	General Boot Error
ROM_BOOT_ERR_UNSUPPORTED	0x1	Unsupported configuration (device/mode selection)
ROM_BOOT_ERR_BMODE_INIT	0x2	Initialization failure for the selected boot mode
ROM_BOOT_ERR_BMODE_CONFIG	0x3	Configuration failure for the selected boot mode
ROM_BOOT_ERR_BMODE_REG_FAILURE	0x5	BMODE registration failure
ROM_BOOT_ERR_SEC_AUTH_FAILURE	0x6	Authentication failure
ROM_BOOT_ERR_SEC_DECRYPT	0x7	Decryption Failure
ROM_BOOT_ERR_ID	0x8	The image being loaded has a lower ID than was previously recorded, indicating image ID is outdated
ROM_BOOT_ERR_CLEANUP	0x9	Cleanup failure
ROM_BOOT_ERR_HDRSGN	0xA	Block header signature failure

Table 40-41: Boot ROM Exceptions (Continued)

Error Name	Error ID	Description
ROM_BOOT_ERR_HOOK	0xB	Hook function failure
ROM_BOOT_ERR_HDRCHK	0xC	Header checksum failure
ROM_BOOT_SECURE_ERR_HDRCHK	0xD	Secure header failure
ROM_BOOT_ERR_NOFINAL	0xE	No final block present
ROM_BOOT_ERR_BLKFILL	0xF	Failure to process fill block
ROM_BOOT_ERR_BLKCRC	0x10	Failed CRC for a block
ROM_BOOT_ERR_DMA	0x12	DMA transfer failure
ROM_BOOT_ERR_CALLBACK	0x13	Callback failure
ROM_BOOT_ERR_NEXTDXE	0x14	Next dxs failure
ROM_BOOT_ERR_PLLCFG	0x15	PLL configuration failure
ROM_BOOT_ERR_INT	0x16	Internal ROM error
ROM_BOOT_ERR_NOKERNEL	0x17	Application returned to the bootrom
ROM_BOOT_ERR_OTPREAD	0x18	Failure to read OTP
ROM_BOOT_ERR_NODEVICE	0x19	No device selected in the bootCommand
ROM_BOOT_ERR_NOBMODE	0x1A	No boot mode selected
ROM_BOOT_ERR_BMODEDIS	0x1B	Boot mode is disabled
ROM_BOOT_ERR_NOPUBKEY	0x1C	No public key available
ROM_BOOT_ERR_NOPVTKEY	0x1D	All private keys have been disabled
ROM_BOOT_ERR_SPICFG	0x1E	Failed SPI configuration
ROM_BOOT_ERR_UARTCFG	0x1F	Failed UART configuration
ROM_BOOT_ERR_SECURITY_FAILURE	0x20	Failure to initialize secure features
ROM_BOOT_ERR_ECDSATYPE	0x21	Wrong ECDSA Type detected other than ECDSA- 224/256
ROM_BOOT_ERR_OTPLOCKBIT	0x22	Error in OTP area holding the security bits
ROM_BOOT_ERR_SH_PARI_IRQ	0x24	L1 Parity Error
ROM_BOOT_ERR_SH_ILOPI_IRQ	0x25	Illegal opcode error
ROM_BOOT_ERR_SH_CB7I_IRQ	0x26	Software stack (Circular Buffer 7) Overflow
ROM_BOOT_ERR_SH_IICDI_IRQ	0x27	Unaligned LW/BW access + unintentional CMMR/SMMR access
ROM_BOOT_ERR_SH_SOVFI_IRQ	0x28	Status loop or mode stack overflow; or PC stack full
ROM_BOOT_ERR_SH_ILADI_IRQ	0x29	Illegal Address Space detected
ROM_BOOT_ERR_SH_TMZHI_IRQ	0x2A	Core Timer error
ROM_BOOT_ERR_SH_BKPI_IRQ	0x2B	User Hardware Breakpoint

Table 40-41: Boot ROM Exceptions (Continued)

Error Name	Error ID	Description
ROM_BOOT_ERR_SH_IIR_IRQ	0x2C	IIR error
ROM_BOOT_ERR_SH_FIR_IRQ	0x2D	FIR error
ROM_BOOT_ERR_SH_SECI_IRQ	0x2E	System event controller interrupt
ROM_BOOT_ERR_SH_RINSEQI_IRQ	0x2F	Restricted Instruction Sequence
ROM_BOOT_ERR_SH_CB15I_IRQ	0x30	Circular Buffer 15 Overflow
ROM_BOOT_ERR_SH_TMZLI_IRQ	0x31	Core Timer (Low Priority Option)
ROM_BOOT_ERR_SH_FIXI_IRQ	0x32	Fixed-point overflow exception
ROM_BOOT_ERR_SH_FLTOI_IRQ	0x33	Floating-point overflow exception
ROM_BOOT_ERR_SH_FLTUI_IRQ	0x34	Floating-point underflow exception
ROM_BOOT_ERR_SH_FLTII_IRQ	0x35	Floating-point invalid exception
ROM_BOOT_ERR_SH_EMULI_IRQ	0x36	Emulator low priority interrupt
ROM_BOOT_ERR_SH_SFT0I_IRQ	0x37	User software interrupt 0
ROM_BOOT_ERR_SH_SFT1I_IRQ	0x38	User software interrupt 1
ROM_BOOT_ERR_SH_SFT2I_IRQ	0x39	User software interrupt 2
ROM_BOOT_ERR_SH_SFT3I_IRQ	0x3A	User software interrupt 3
ROM_BOOT_ERR_OTPBOOT_IRQ	0x3B	OTP Boot exception

Boot ROM Programming Model

This section describes the programming model for booting the processor. The programming model includes booting functions, API calls, and data structures.

Boot Mode Driver API

The kernel provides a mechanism to provide a customization of supported boot modes or for implementation of completely new boot modes as second stage boot loaders. This allows programs to customize booting while still taking advantage of the rest of the booting framework. A custom boot mode may provide support for a peripheral that is not supported for boot by the ROM, or it could support one of the same peripherals but with a different configuration.

All the same security features can be supported when using a custom boot mode.

A full boot mode, as perceived by the boot implementation, is a collection of following functions:

1. Register – installs the driver functions listed below so they can be accessed by the boot process
2. Initialization – initialize the boot source
3. Configuration – configure the boot source

4. Load – read from the boot source
5. Cleanup – called after booting

The boot kernel is only aware of functions 2 through 5 and has a requirement to support the Load function. It is this function that is responsible for fetching the boot stream from the boot peripheral. The other functions are used prior to executing the kernel or for cleaning up after the kernel has completed processing the boot stream.

To install a custom boot mode:

1. Create a first stage boot application to define a Load function
2. Use the `adi_rom_BootKernel()` API to call the boot kernel once the boot peripheral and pinmuxing has configured. Ensure all the fields of `struct ADI_ROM_BOOT_CONFIG` are configured accordingly prior to performing the call.

The boot mode can use the `pModeData` member of `ADI_ROM_BOOT_CONFIG` to preserve and access shared data across the different function calls if required.

All functions have the following prototype:

```
void apiFunction(ADI_ROM_BOOT_CONFIG* pBootStruct);
```

Another way to support custom boot mode is:

1. Create a first stage boot application to define all Init, Config, Load and Cleanup routines.
2. Use `adi_rom_Boot()` API with hook function installed to update the Init, Config, Load and Clean up functions after the preregister initialization is complete. This is checked by boot ROM to override the above functions inside the `ROM_HOOK_CALL_CAUSE: ROM_HOOK_REG_COMPLETE`

Load Function

The load function is required to read data from the source into the specified destination, according to the parameters given through the configuration `struct` parameter. The structure provides all of the required information read from the block header, or specified by the kernel to read the block header. The load function often makes use of the DMA APIs in order to simplify the load function implementation.

As the kernel processes the stream, it calls the load function to request data. Initially, the request is for the header, then the kernel makes requests according to the block flags it parses. The load function must only read from the device, and write where requested.

Relevant fields within the `ADI_ROM_BOOT_CONFIG` object for the load function are (but not limited to): `uwDataWidth`, `pSource`, `dByteCount`, `pDestination`, `loadType`.

Custom load functions must meet the following requirements.

- Protect against `dByteCount` values of zero
- Use multiple DMA units if `dByteCount` is greater than 65536 and the peripheral does not support byte count transfers greater than 65536

- The `pSource` and `pDestination` pointers must be properly updated after loading.

In slave boot modes, the boot kernel uses the address of the `dArgument` field in the `pHeader` block as the destination for the required dummy DMAs when payload data is consumed from `ROM_BFLAG_IGNORE` blocks. The load function must read the `ARGUMENT` word of the block early in the function (if required).

Initialization/Configuration Function

The Initialization and Configuration functions are called in sequence when calling a boot operation using an already supported boot peripheral via the `adi_rom_Boot()` API. These functions are used to configure the boot peripheral prior to calling the boot kernel. Both functions are called in sequence separated only by a call to a user-defined hook function. This hook function is useful when using built-in boot modes to further customize their function. The initialization and configuration functions are responsible for applying any required settings to any devices in use. For example, pin multiplexing may be required and data or pointers that are used by the load function must be initialized. The specific actions depend on the device and functionality used.

Cleanup Function

The cleanup function is called after the entire boot stream is read, and the kernel has completed its boot mode-specific function. This is only performed when using the `adi_rom_Boot()` API. Resetting any status registers, or device parameters is done to prepare the environment for the execution of the newly loaded application.

Error Handler

This section describes the default error handler for the ROM including information on how to customize the error handling.

The default error handler eventually puts the core into an idle state. This functionality can be overridden by using an Init Block (see [Block Types](#)) to modify the error function point in the `struct ADI_ROM_BOOT_CONFIG` structure. The error handler has access to the entire boot info structure and receives the instruction address that triggered the error.

When a part is locked, and the boot type has not disabled secure boot, only the default error handler is called.

The expected prototype is:

```
void ErrorFunction(ADI_ROM_BOOT_CONFIG* pBootStruct, void *pFailingAddress);
```

The error handler saves the failing address to the `ADI_ROM_BOOT_CONFIG` structure then raises the `INTR_SOFT3` fault signaling a fault condition to the system before then entering an endless loop in the boot rom.

NOTE: When using the `adi_rom_Boot()` function to perform a boot action, programs may need to manually configure the `INTR_SOFT3` fault signaling depending on the previous application software executed. Calling the boot process using `adi_rom_Boot()` does not result in a reset of the SEC and Fault installation and configuration as described in [Preboot Operations](#).

Page Mode

Page operations are beneficial for page oriented boot source devices, and to improve boot performance for secure boot operations. Page mode optimizes memory reads for block organized devices by always reading a page, rather than reading data on demand. Two 1024 byte buffers are used in page mode where the contents of one buffer are processed by the boot kernel while DMA is used to load the next data into the second buffer.

Blocking DMA is used to load the active buffer, forcing the process to pause until the DMA is complete. Non-blocking DMA is used to load the not active buffer, allowing the active buffer to process while loading the new data in parallel.

Page mode can be enabled when calling a boot mode using `adi_rom_Boot()`. Refer to the API documentation for the various modes supported by this API. Additionally programs can set the flag using the `dFlags` function (see [struct ADI_ROM_BOOT_CONFIG](#)) when using hook functions

NOTE: Do not customize page mode settings from the default installed by the boot process.

Boot Hook Function

The boot software allows installation of callback hooks through the use of the `adi_rom_Boot()` APIs hook function parameter. By using this feature, it is possible to alter the state of the processor, at different stages of the boot process and customize the boot structures to alter the behavior of the boot process.

The hook function must adhere to the following prototype:

```
nt32_t
hookFunction(ADI_ROM_BOOT_CONFIG* pBootconfig,
ROM_HOOK_CALL_CAUSE cause);
```

By modifying settings in the `ADI_ROM_BOOT_CONFIG` structure, many alterations of the boot process can be achieved. Much of the same functionality that is available in an Init Block can be provided through the hook function, with even more flexibility for customization. The hook function is called once after executing the boot modes Init routine then once again after executing the boot modes Config routine. A flag passed to the hook function allows software to determine at which point the call took place to allow for conditional processing to occur at different stages of the setup phase.

The hook function must return a zero value for normal booting to continue. A non-zero return value causes the ROM to omit loading any data and immediately transfer control according to [Boot Termination and Application Execution](#).

When the hook function is called, a parameter is passed that indicates why the hook function was called. See [enum ROM_HOOK_CALL_CAUSE](#) for more details.

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

An optional hook routine is provided as a callback when calling a boot mode via `adi_rom_Boot`. This hook routine is called by the boot software first after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 40-42: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
ROM_HOOK_CALL_INIT_COMPLETE	Call was as a result of completion of the boot modes initialization function
ROM_HOOK_CALL_CONFIG_COMPLETE	Call was as a result of the completion of the boot modes configuration function
ROM_HOOK_REG_COMPLETE	Call was as a result of the completion of the boot modes pre-register initialization

Boot Return Feature

The `adi_rom_Boot()` API provides a feature to bypass calling the loaded application when boot completes, and to return to the routine that made the call instead. The boot software returns the next address after the last loaded application block in the boot source when this feature is enabled.

To enable this feature, set the `ROM_BFLAG_RETURN` flag in the `adi_rom_Boot()` `flags` argument when calling the API.

Boot Termination and Application Execution

When the boot kernel completes the processing of the boot stream, a sequence of events is required to then pass control to the loaded application.

When the boot process is complete, the core is required to vector to the application start address and start executing the newly loaded application. Typically, the first block of a boot stream, which is marked with the `BFLAG_FIRST` flag, contains the address of the application. The application entry point for each core is loaded into the cores corresponding `RCU_SVECTn` register.

Upon boot completion only the core that performed the boot process will vector and start executing the loaded application. This core must then manage the process of resetting then releasing from the reset the other cores in the system in order to make them execute their newly loaded applications.

Execution of the loaded application can be bypassed when calling the boot mode using `adi_rom_Boot()` and setting the `ROM_BFLAG_RETURN` flag.

Table 40-43: Application Entry Point Registers

Core ID	Corresponding RCU_SVECTn Register
0	<code>RCU_SVECT0</code>

Boot ROM OTP Customizations

The boot ROM provides a mechanism through available non-volatile programmable memory (OTP on this processor) to customize different aspects of the boot process. These customizations include: overriding default boot-peripheral instance, overriding default peripheral-timing parameters and disabling boot modes.

Data in OTP memory controls all ROM customization. The `struct ADI_ROM_OTP_BOOT_INFO` data structure accounts for most of the options.

CGU Initializations

Refer to [CGU Configuration](#)

Boot Command Customization

Refer to [Boot Command Customization](#)

DMC Configuration

Refer to [DMC Configuration](#)

Secure Boot Customization

All the public and private keys can be invalidated using the various key invalidation fields provided in the `ADI_ROM_OTP_BOOT_INFO` structure. This configuration is useful when a new key is required. The boot ROM always uses the lowest valid key enumeration. If key0 is valid, then it is used, if key0 is invalid and key1 is valid, then key1 is used. Refer to [Secure Boot](#) for details on the secure boot functionality.

Disabling Boot Modes

Refer to [Boot Mode Disable](#)

API Reference

The APIs defined in this section are exposed for general use.

`adi_rom_Boot()`

Provides access to boot an application at run-time through a supported peripheral.

API Details

```
void * adi_rom_Boot(
    void * pAddress,
    uint32_t flags,
    int32_t blockCount,
    ROM_BOOT_HOOK_FUNC * pHook,
    uint32_t command
)
```

pAddress

Pointer to source address of the boot stream.

flags

Global flags applied to the entire boot process

blockCount

Number of blocks to boot. Zero results in processing until a final block is reached.

pHook

Pointer to user implemented hook function for enabling callbacks during the registering of the boot mode with the boot kernel

command

The boot command defining the boot mode to use, the peripheral instance to boot from as well as some boot mode specific configuration

Returns

The 32-bit address of the next address in the boot source to process

Function Description

This function is used for any kind of second-stage boot for an already supported boot mode. It provides options to boot from any peripheral enumeration. In SPI Master boot the function supports using any SPI slave select signal.

Boot modes may support an auto-detection mechanism to identify the type of connected device. This function provides options to bypass such auto-detection and use custom configuration options. Options are also provided to bypass peripheral configurations such as pinmux settings or a peripheral configuration if an existing peripheral is more appropriately configured to allow communication with the boot source.

These features are all provided via the command parameter which is specific for each particular boot mode.

The source address of the boot stream is required for master boot modes that require that an address is issued in order to request data from the boot source. Slave boot modes are under full control of the host and use a handshake mechanism to indicate that the processor is ready to receive data. For boot modes such as UART Slave and SPI Slave this parameter is of little value in regards to the boot process itself. However it can prove useful in debug to see how far through the boot stream the boot process progressed in the event of a boot failure.

NOTE: The processor supports both SPI Memory-Mapped boot as well as Peripheral based SPI Boot. When the boot mode is called to boot from the memory-mapped boot mode via the command argument, the address must coincide with the processors memory-mapped SPI address space as defined by the processors internal memory map. When using the peripheral based boot mode use the absolute address of the boot stream in flash.

Flags passed using the `flags` argument are *global* flags and the operation is applied throughout the entire boot process. These must not be confused with the boot *block specific* flags which are part of the boot stream and indicate how a particular block in the boot stream is processed. Internally, the boot kernel takes the global flags supplied using this function call and combines them with a boot block's local flags to determine all the operations to perform on a given block. After processing the boot block the local flags are cleared and are ready to populate from the next boot block while the global flags remain.

Table 40-44: Global Flags

Bit Position	Flag Name	Description
18	ROM_BFLAG_HOOK	Calls the application supplied hook function after execution of bootmode init and config routines
19	ROM_BFLAG_PAGEMODE	Enables page mode processing where blocks of data are fetched and processed from internal memory
20	ROM_BFLAG_NOFIRSTHEADER	Set if calling the boot mode and the first block header has already been fetched and is present in the block header storage location
21	ROM_BFLAG_HEADER	Not set by the application, set by the boot code each time it is fetched by a block header
22–25	Reserved	
26	ROM_BFLAG_SLAVE	Slave boot mode. Causes different handling of ignore blocks by the kernel
27	ROM_BFLAG_WAKEUP	Enables conditional processing of boot blocks intended for wakeup events but not exclusively
28	ROM_BFLAG_NEXTDXE	Parse stream via next DXE pointer
29	ROM_BFLAG_RETURN	Returns the application after calling the API instead of running the new application
30	Reserved	
31	ROM_BFLAG_NORESTORE	Do not execute the boot peripherals cleanup routine to restore register contents

The `blockCount` argument specifies the number of blocks to process before terminating the boot process (default = 0x00000000). This value instructs the boot software to continue processing a boot stream until the `ROM_BFLAG_FINAL` flag is set. To load only a specified number of blocks, programs can use `blockCount` argument.

When `blockCount` is used in combination with the `ROM_BFLAG_NEXTDXE` flag then the block count is repurposed as a next application count. The boot kernel navigates the first blocks of multiple boot streams similar to a linked list. When the requested application count is reached, the program returns the pointer to this application in the boot source. This allows programs to use the boot kernel to find a specific application when multiple application boot streams are stored contiguously in the boot source.

The `pHook` argument is a function pointer to a hook routine. When used with the `ROM_BFLAG_HOOK` global flag the boot mode calls the hook routine after calling the boot modes `init` and functions in the boot mode's driver so that the boot software can be customized and the configuration changed.

The `pHook` argument when used with the `ROM_BFLAG_HOOK` global flag can enable new features not supported by the boot software and to install a custom load function or error handler for example.

The `command` argument describes the boot peripheral to boot from, the peripheral instance and contains additional boot mode specific configuration information and flags specific to the boot mode.

NOTE: When calling a boot mode via this API the program must first ensure that the boot peripheral is configured in the SPU as a secure master. The boot software does not configure the peripheral security via this API so that device security is fully controlled by a dedicated task.

adi_rom_BootKernel()

Calls the boot kernel allowing for implementation of custom boot modes.

API Details

```
void * adi_rom_BootKernel (ADI_ROM_BOOT_CONFIG * pBoot)
```

pBoot

Pointer to the `struct ADI_ROM_BOOT_CONFIG` boot structure containing the complete context of the boot configuration.

Returns

Pointer containing the address of the byte immediately following the end of the boot stream.

Function Description

The boot kernel performs the core processing of the boot stream. The boot kernel calls a load function to load data from the peripheral to the required destination. The boot kernel itself has no concept of what the boot peripheral is or how that peripheral is configured. The kernel calls the registered load function and the load function must then analyze the boot structure and provide the requested amount of data to the required destination.

The load function called by the kernel is provided via the `struct ADI_ROM_BOOT_REGISTRY` member of `struct ADI_ROM_BOOT_CONFIG`.

The boot kernel fetches a boot stream block header then a payload if one exists. The boot kernel takes care of the size of the data being requested and the destination address.

The load function that is registered with the kernel is required to update the `pSource`. Keeping this control under the load function as opposed to the boot kernel itself allows load functions to better control where the next block of data is fetched in the event the boot stream is fragmented or split into different areas of the boot source.

This function is used to implement a second stage boot loader for a peripheral in which there is no driver support in the boot ROM. The application is responsible for initializing the peripheral and the complete `struct ADI_ROM_BOOT_CONFIG` object before calling this function. The application is responsible for performing a vector to the newly loaded application on return from the function.

NOTE: Programs must ensure that when a new application is loaded it does not clobber the load function and the part of the software responsible for making the core jump to the start of the newly loaded application.

adi_rom_Crc32Init()

The CRC32 Initcode function in the boot rom that is called to enable CRC32 support of boot stream payloads.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Init(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the `struct ADI_ROM_BOOT_CONFIG` object containing the complete boot configuration

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_CRC_INITCODE_ERR` when `pBootConfig` or `pBootConfig->pHeader` are zero
- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` when the callback is registered and lookup table initialized

Function Description

The boot process supports CRC32 protection of all boot block payloads. In order to enable this feature a global callback must be registered with the boot process using `struct ADI_ROM_BOOT_CONFIG` and the CRC peripherals look up table initialized from the application's polynomial.

An init block header where the `ROM_BFLAG_INIT` flag is set must be included in the boot stream to enable CRC functions. The `ADI_ROM_BOOT_HEADER::pTargetAddress` field must be set to the address of this function and the application polynomial is provided using the block's `struct ADI_ROM_BOOT_HEADER` member.

When the boot kernel processes the init block it calls this function in the boot ROM and registers the callback with the kernel and performs the look up table initialization.

CRC functionality is enabled on MDMA channel 1 interfaced to the CRC0 peripheral instance.

adi_rom_Crc32Poly()

Initializes the CRC peripheral for use with the user supplied polynomial.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Poly(
    uint32_t CrcPoly,
    ROM_BOOT_MDMA_REGS const *const pDma
)
```

CrcPoly

None

pDma

None

Function Description

The CRC lookup table must be initialized for the CRC polynomial of choice to prepare the CRC peripheral for use.

adi_rom_GetAddress()

Used to find the location of various look-up tables and data objects used during the boot process.

API Details

```
int32_t adi_rom_GetAddress(ROM_GETADDR_VALUE value)
```

value

The [enum ROM_GETADDR_VALUE](#) enumeration specifying the object to retrieve the address of in the ROM memory

Returns

The byte address of the object in memory

Function Description

The function returns the address of the object specified by the enumerator provided as an argument to the function. Using this function can make software more code compatible with future products and silicon revisions.

adi_rom_MemCompare()

Verifies that a block of data is filled with a application supplied 32-bit value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCompare(
    ROM_DMA_MDMA_CONFIG * pDmaCfg,
    ROM_BOOT_MDMA_REGS const *const pDma
)
```

pDmaCfg

None

pDma

None

Function Description

The CRC peripheral is used in compare mode and a source MDMA channel is used to read data from a buffer and supply each 32-bit value to the CRC. The CRC peripheral checks that the incoming 32-bit value matches the 32-bit value to compare against.

adi_rom_MemCopy()

Performs a Memory-to-Memory DMA (MDMA) operation using a source and destination pair of DMA channels.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCopy (
    ROM_DMA_MDMA_CONFIG * pDmaCfg,
    ROM_BOOT_MDMA_REGS const *const pDma
)
```

pDmaCfg

Pointer to the `struct ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

pDma

Pointer to the `struct ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channel MMRs and associated CRC peripheral

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation is performed
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- if a configuration error occurred in the source MDMA channel
- if a configuration error occurred in the destination MDMA channel

Function Description

The memory copy routine performs transfers of blocks of data from one memory location to another. The routine takes a basic descriptor providing configuration details of the operation to perform using the `struct`

`ROM_DMA_MDMA_CONFIG` object passed as the first argument. The second argument is a descriptor that provides access to the DMA channel's MMR registers and the associated CRC peripheral. When called from the higher level `adi_rom_MemDma()` routine this object is retrieved from the ROM.

NOTE: Applications are expected to make use the `adi_rom_MemDma()` routine for all MDMA operations as there is little additional optional configuration that is supported by using this routine.

adi_rom_MemCrc()

Performs CRC32 verification of a block of data by reading the contents and comparing them with an expected result.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCrc (
    ROM_DMA_MDMA_CONFIG * pDmaCfg,
    ROM_BOOT_MDMA_REGS const *const pDma
)
```

pDmaCfg

Pointer to the `struct ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

pDma

Pointer to the `struct ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channel's MMRs and associated CRC peripheral

Function Description

The routine uses an MDMA channel pair source DMA channel and the CRC peripheral to calculate a CRC32 result of a data block using a previously supplied polynomial.

The polynomial is supplied through the `adi_rom_Crc32Poly()` routine to ensure consistent CRC peripheral configuration for the look up table initialization that uses the polynomial and the

NOTE: Applications are expected to make use the `adi_rom_MemDma()` routine for all MDMA operations, there is little additional optional configuration that is supported by using this routine.

adi_rom_MemDma()

Provides access to all MDMA operations supported by the boot ROM implementation.

API Details

```
ROM_BOOT_RESULT adi_rom_MemDma (ROM_DMA_MDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the `struct ROM_DMA_PDMA_CONFIG` object containing the MDMA configuration

Returns

Returns the following results using the enum `ROM_BOOT_RESULT` enumeration.

- `ROM_BOOT_SUCCESS`. Successful operation or when byte count is 0 as no operation is performed
- `ROM_BOOT_MDMA_ID_ERR`. A MDMA channel ID is provided that is not supported
- `ROM_BOOT_CRC_SUPPORTED_ERR`. A CRC operation was attempted on a MDMA channel that does not support CRC
- `ROM_BOOT_MDMA_OPERATION_ERR`. The operation is not supported
- `ROM_BOOT_DMA_FAILURE`. A configuration error is detected in the DMA channel prior to configuring the channel for the new operation
- `ROM_BOOT_DMA_FAILURE`. A configuration error was detected in the DMA channel and the operation requested involves only a single DMA channel
- A configuration error occurred in the source MDMA channel
- A configuration error occurred in the destination MDMA channel
- `ROM_BOOT_CRC_COUNT_ERR`. A CRC operation is being requested and the byte count is not a multiple of 4
- `ROM_BOOT_CRC_FAILURE`. A CRC32 verification fails
- `ROM_BOOT_CRC_FAILURE`. A 32-bit memory compare fails

Function Description

The supported MDMA operations are listed in enum `ROM_DMA_MDMA_OPERATION`.

NOTE: The MDMA and CRC peripherals support on-the-fly CRC32 calculations during the transfer of data from one location to another—MDMA in the boot ROM software does not. For CRC calculations data is instead read back from its final destination and verified.

This function is the main entry point for all the MDMA functions supported by the boot ROM software. The individual functions that are called for each operation type are also exposed using the public API.

NOTE: Use this function for all operations. The lower level functions allow for some basic reconfiguration of default parameters but such modifications are not be required where these basic MDMA operations are required.

The boot ROM has an MDMA configuration data structure that is used to specify the overall MDMA configuration of the processor. It provides details on the DMA channel ID associated with each MDMA channel's source and destination DMA channel. It provides information about CRC support and the CRC peripheral instance that is used for a given MDMA channel. See `struct ROM_BOOT_MDMA` and `struct ROM_BOOT_MDMA_REGS` for full details of the content stored.

The configuration provided as the only argument to the function is provided in [Table 40-64 ROM_DMA_MDMA_CONFIG Members](#).

For a basic MDMA transfer from source to destination the program needs to configure:

- The `struct ROM_DMA_MDMA_CONFIG` type as `enum ROM_DMA_MDMA_OPERATION`
- The MDMA channel to use via `eId` MDMA Channel ID for example
- Set the address of the source data via `pSource` Source Pointer
- Set the destination address of the source data via `pDestination` Destination Pointer
- Set the byte count via `ByteCount` Byte Count
- Set `eDoneDetect` DMA Done Detection Method to `enum ROM_DMA_DONE_DETECT_METHOD` in order to poll for DMA completion

NOTE: The implementation of the MDMA operations does not support interrupt driven data transfers. The routines are implemented for polling DMA status during the boot process. The boot stream is restricted where byte counts and source and destination address alignment must all be a multiple of 4 bytes. MDMA routines must comply with these restrictions.

To use the CRC32 function of the MDMA routines, the application must first initialize the CRC lookup table from the application supplied polynomial. This operation can be performed by setting `ROM_DMA_MDMA_CONFIG::eOperation` type as `ROM_DMA_MDMA_OPERATION::ROM_DMA_CRC_LUT_INIT`. If an MDMA channel is specified that does not support CRC function, an error result is returned.

For further details of the individual operations supported see the following API references:

- [adi_rom_MemCopy\(\)](#)
- [adi_rom_MemCrc\(\)](#)
- [adi_rom_MemFill\(\)](#)
- [adi_rom_MemCompare\(\)](#)
- [adi_rom_Crc32Poly\(\)](#)

adi_rom_MemFill()

Fills a block of memory with a 32-bit user supplied value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemFill(
    ROM_DMA_MDMA_CONFIG * pDmaCfg,
    ROM_BOOT_MDMA_REGS const *const pDma
)
```

pDmaCfg

None

pDma

None

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation to be performed
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel prior to configuring the channel for the new operation
- A configuration error occurred in the source MDMA channel
- A configuration error occurred in the destination MDMA channel

Function Description

The CRC peripheral is configured for fill mode and the destination MDMA channel is configured to fill a block of memory with a fixed 32-bit value.

adi_rom_PeriphDma()

Provides access to any peripherals dedicated DMA channel for receive operations only.

API Details

```
ROM_BOOT_RESULT adi_rom_PeriphDma(ROM_DMA_PDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the `struct ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

Returns

Returns the following results

- `ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation is performed
- `ROM_BOOT_DMA_ACTIVE` if the DMA channel is currently running
- `ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel after starting the DMA operation

Function Description

The peripheral DMA routine is used by the load routines of boot peripherals that have dedicated DMA channels and do not support MDMA channel pairs. Examples are the SPI when not configured for memory-mapped mode and UART peripherals.

In the boot implementation this routine is called from the peripheral load function to request data from the boot source. The routine supports both polling on DMA completion and non-blocking operation to allow for immediate return after starting the DMA operation and continuing with further processing.

NOTE: The function only supports read operations from the peripheral to memory. Transmit operations from memory to peripheral are not supported

adi_rom_otp_cfg()

Configures the OTPC to enable read and program operations.

API Details

```
bool adi_rom_otp_cfg(void)
```

Function Description

Programs may call this routine to ensure the OTPC is configured correctly for read and write access.

NOTE: The preboot process configures the OTPC for use and there is no direct requirement to call this function when using the OTP.

adi_rom_otp_get()

Reads the field from OTP as defined by the supplied [enum OTPCMD](#).

API Details

```
bool adi_rom_otp_get (
    OTPCMD cmd,
    uint32_t data[]
)
```

cmd

The [enum OTPCMD](#) enumeration describes the OTP content to read

data[]

Pointer to storage area to store the read OTP contents

Function Description

Programs can read the various fields ([Table 40-69 OTPCMD Members](#)) of the OTP via this routine. The supplied `enum OTPCMD` object is used to specify the object to read.

adi_rom_otp_lock()

Locks the processor, enabling all security features.

API Details

```
bool adi_rom_otp_lock(void)
```

Function Description

This function is used to lock the processor from unauthorized access. Once this function is called the application must supply a secure debug key in order to gain access to the device with debug tools and the processor may only be booted using a secure boot stream.

WARNING: Programs must ensure that the OTP secure boot fields are all programmed. Secure boot can be verified prior to locking the processor. Programs must also provision a secure debug key.

adi_rom_otp_pgm()

Programs the OTP Memory with the contents of the `struct otp_data` object.

API Details

```
bool adi_rom_otp_pgm(otp_data * data)
```

data

Pointer to the `::otp_data` object containing the complete OTP contents to program

Function Description

The OTP memory is only programmed with values that are not 0. Any items that are 0 are ignored. Programs are expected to use this function for all OTP program operations.

callback()

Implements custom callbacks to previously loaded code during boot.

API Details

```
ROM_BOOT_RESULT callback(
    ADI_ROM_BOOT_CONFIG * pBootConfig,
    ADI_ROM_BOOT_BUFFER * pBuffer,
    uint32_t nFlags
)
```

pBootConfig

Pointer to the `struct ADI_ROM_BOOT_CONFIG` object containing the complete context of the boot procedure

pBuffer

Pointer to the `struct ADI_ROM_BOOT_BUFFER` object containing details of the payload associated with the callback

nFlags

The callback flags as set by the boot kernel

Function Description

A single callback function may be registered with the boot kernel using `pCallbackFunction` Pointer to the callback function that is called when processing boot blocks with the callback flag set . This function is then called whenever a block is processed with the `ROM_BFLAG_CALLBACK` flag set. Only a single callback function can be registered for the complete boot process.

Callbacks may be used alongside indirect blocks when post processing of the received boot data is required before sending the data to the final destination. An example of this is compression applied to block payloads. The compressed payload is loaded indirectly to the intermediate buffer where it is decompressed by the callback. The callback can modify the source address and byte count for the final MDMA transfer of the decompressed payload using the `pBuffer` parameter. The callback returns the boot kernel then handles the final transfer of the uncompressed data to its destination.

There are restrictions on the amount of data that can be loaded using indirect blocks depending on the size of the intermediate buffer. For this reason the `nFlags` parameter is used to indicate the status of the callback when handling larger blocks of indirect data. The table below defines the supported flags:

Bit Position	Flag Name	Description
0	<code>ROM_CBFLAG_DIRECT</code>	When set indicates the call was from the processing of a block header with the <code>ROM_BFLAG_CALLBACK</code> flag set
1	<code>ROM_CBLAG_PAGESTART</code>	Indicates the callback was a result of a fetch of a page of data to the intermediate buffers
2	<code>ROM_CBFLAG_FIRST</code>	Set if the first fetch of payload data
3	<code>ROM_CBFLAG_FINAL</code>	Set if the final fetch of payload data
31:4	Reserved	Reserved

When a callback block header is received by the boot kernel, a call to the callback is performed with the `ROM_CBFLAG_DIRECT` flag set. If the `ROM_BFLAG_INDIRECT` flag or the `ROM_BFLAG_PAGEMODE` flags are set they indicate the use of indirect or page mode the `ROM_CBFLAG_FIRST` and `ROM_CBFLAG_FINAL`

flags are cleared. If the transfer is a direct transfer straight to the final destination and not via the intermediate buffers, then the ROM_CBFLAG_FIRST and ROM_CBFLAG_FINAL flags are also set.

This allows software to identify a callback call based on the processing of a block header with the ROM_BFLAG_CALLBACK flag set.

Callbacks are also called when processing payloads indirectly or when page mode is enabled.

- The ROM_CBFLAG_DIRECT flag is cleared when the callback is a result of processing the payload data using the intermediate buffers.
- The ROM_CBFLAG_FIRST flag is set if the callback is a result of fetching the first block of data in the payload. This flag is also set if the complete block of data fits in the intermediate buffer.
- If the payload does not fit completely in the intermediate buffers multiple fetches must take place and thus multiple callbacks are generated.
- If no flags are set it indicates a callback on a payload transfer is neither the first nor the last block of data in the payload, so there is still further data in the payload to fetch.
- If only the ROM_CBFLAG_FINAL flag is set then it is the final block in a payload transfer.

The following table provides an overview of the flag states and their meaning for the processing of callbacks.

ROM_CBFLAG_DIRECT	ROM_CBFLAG_PAGESTART	ROM_CBFLAG_FIRST	ROM_CBFLAG_FINAL	Description
1	0	0	0	Callback as a result of processing a block header with indirect or page-mode enabled
1	0	1	1	Callback as a result of processing a block header with indirect and pagemode disabled
0	1	0	0	Callback as a result of fetching a page of data in pagemode
0	1	0	1	Callback as a result of fetching a page of data in pagemode and the final page in the block
0	0	1	0	Callback as a result of fetching the first part of payload in an indirect payload
0	0	0	0	Callback as a result of fetching an indirect payload, not first or last transfer in payload
0	0	0	1	Callback as a result of fetching the final part of payload in an indirect payload

ROM_CBFLAG_DIRECT	ROM_CBFLAG_PAGESTART	ROM_CBFLAG_FIRST	ROM_CBFLAG_FINAL	Description
0	0	1	1	Callback as a result of fetching the complete payload in an indirect payload

initcode()

Implements custom callbacks to previously loaded code during boot.

API Details

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the `struct ADI_ROM_BOOT_CONFIG` object containing the complete context of the boot procedure

Function Description

Initcode functions are embedded into the boot stream to execute application code during the boot phase. Initcode functions help to optimally configure the CGU or any external memory interface that requires initialization to boot data to those memories.

A boot stream may have any number of initcodes present. The only requirement is that the code must be loaded prior to processing the `BFLAG_INIT` block.

The initcode routine is passed by the pointer to the complete boot context so that extensive boot customization tasks can be supported.

adi_rom_idle_loop()

Jumps to the IDLE loop inside the boot ROM space.

API Details

```
void adi_rom_idle_loop(void)
```

Function Description

This function is used to jump to the forever idle loop inside the boot ROM space.

adi_rom_CguInit()

Can be used for CGU initialization.

API Details

```
bool adi_rom_CguInit(const ADI_ROM_OTP_BOOT_CGU_INFO * const pSettings)
```

pSettings

Pointer to the struct `struct ADI_ROM_OTP_BOOT_CGU_INFO` object containing complete context of CGU initialization

Function Description

This function is used for CGU initialization.

adi_rom_DmcPhyCalibration()

Used for DMC PHY calibration.

API Details

```
void adi_rom_DmcPhyCalibration(ADI_ROM_OTP_DMC_CONFIG *pConfig, int csel_dsel_r)
```

pConfig

Pointer to the struct `struct ADI_ROM_OTP_DMC_CONFIG` object containing the complete context of the DMC PHY calibration

csel_dsel_r

Holds the CSEL to DSEL ratio

Function Description

This function is used for DMC PHY calibration.

adi_rom_DmcInit()

Used for DMC initialization.

API Details

```
void adi_rom_DmcInit(ADI_ROM_OTP_DMC_CONFIG *pConfig, uint32_t csel_dsel_r)
```

pConfig

Pointer to the struct `struct ADI_ROM_OTP_DMC_CONFIG` object containing the complete context of the DMC PHY calibration.

csel_dsel_r

Holds the CSEL to DSEL ratio.

Function Description

This function is used for DMC PHY calibration.

adi_rom_Shalnit()

Initializes the PKTE module for the SHA-224/256 operation.

API Details

```
void adi_rom_DmcInit(ADI_ROM_OTP_DMC_CONFIG *pConfig, uint32_t csel_dsel_r)
```

pCrypto

Pointer to Crypto descriptors

IV

Initialization Vector

Ecdsa_Word

ECDSA word count for 224/256 bit operation

Function Description

This API is used for the PKTE initialization for the SHA-224/256 bit operation

adi_rom_Sha()

Computes SHA-224/256 bit digest from the PKTE module.

API Details

```
adi_rom_Sha(CRYPTO_DESCRIPTOR * pCrypto, uint32_t Ecdsa_Word, void *output_p, void *input_p, int size, bool finalFlag)
```

pCrypto

Pointer to Crypto descriptors

Ecdsa_Word

ECDSA word count for 224/256 bit operation

output_p

Pointer to the output buffer

input_p

Pointer to the input buffer

size

Size of the input buffer. It must be a multiple of 512

finalFlag

Final block or not

Function Description

This API is used to compute the message digest using SHA-224/256 using the PKTE module.

/* points to bootrom_jumtable_aes128_cbc_decrypt */

```

        typedef ROM_BOOT_RESULT (*_bootrom_jumtable_aes128_cbc_decrypt_t)
(CRYPTO_DESCRIPTOR
        * pCrypto, void *output_p, void *input_p, int size);
        inline static ROM_BOOT_RESULT
adi_rom_Aes128CbcDecrypt(CRYPTO_DESCRIPTOR * pCrypto,
        void *output_p, void *input_p, int size){
        _bootrom_jumtable_aes128_cbc_decrypt_t pTmp =
        (_bootrom_jumtable_aes128_cbc_decrypt_t)
(FUNC_ROM_AES128_CBC_DECRYPT);
        return (*pTmp)(pCrypto, output_p, input_p, size);
        )

```

/* points to bootrom_jumtable_aes128_key_unwrap */

```

        typedef ROM_BOOT_RESULT (*_bootrom_jumtable_aes128_key_unwrap_t)
(CRYPTO_DESCRIPTOR
        * pCrypto, void *WrappedIn_p, void *KeyOut_p, uint32_t KeyBits,
void *KWK_p);
        inline static ROM_BOOT_RESULT
adi_rom_Aes128KeyUnwrap(CRYPTO_DESCRIPTOR * pCrypto,
        void *WrappedIn_p, void *KeyOut_p, uint32_t KeyBits, void *KWK_p){
        _bootrom_jumtable_aes128_key_unwrap_t pTmp =
        (_bootrom_jumtable_aes128_key_unwrap_t)
(FUNC_ROM_AES128_KEY_UNWRAP);
        return (*pTmp)(pCrypto, WrappedIn_p, KeyOut_p, KeyBits, KWK_p);
        }

```

/* points to bootrom_jumtable_aes128_cbc_loadkey */

```

        typedef void * (*_bootrom_jumtable_aes128_cbc_loadkey_t)(sa_t *
pSARRecord, uint32_t
        *key_p);
        inline static void * adi_rom_Aes128CbcLoadkey(sa_t * pSARRecord,
uint32_t *key_p)
        {
        _bootrom_jumtable_aes128_cbc_loadkey_t pTmp =
        (_bootrom_jumtable_aes128_cbc_loadkey_t)

```



```
(FUNC_ROM_AES128_CBC_LOADKEY);
    return (*pTmp)(pSARRecord, key_p);
}
```

/* points to bootrom_jumtable_pka_init */

```
typedef PKA_Status_t (*_bootrom_jumtable_pka_init_t)( PKA_IOArea_t *
const IOArea_p,
    Device_Handle_t Device, const uint32_t * const Firmware_p, const
uint32_t
    FirmwareWordCount);
inline static PKA_Status_t adi_rom_PKA_Init( PKA_IOArea_t * const
IOArea_p,
    Device_Handle_t Device, const uint32_t * const Firmware_p, const
uint32_t
    FirmwareWordCount) {
    _bootrom_jumtable_pka_init_t pTmp =
        (_bootrom_jumtable_pka_init_t)(FUNC_ROM_PKA_INIT);
    return (*pTmp)(IOArea_p, Device, Firmware_p, FirmwareWordCount);
}
```

/* points to bootrom_jumtable_ecdsa_verify_init */

```
typedef ROM_BOOT_RESULT
(*_bootrom_jumtable_ecdsa_verify_init_t)(SBHYBRID_EcdsaContext_t
* const Verify_p,
    const SBIF_ECDSA_PublicKey_t * const PublicKey_p, const
SBIF_ECDSA_Signature_t *
    const Signature_p, uint32_t EcdsaWord);
inline static ROM_BOOT_RESULT
adi_rom_EcdsaVerifyInit(SBHYBRID_EcdsaContext_t * const
    Verify_p, const SBIF_ECDSA_PublicKey_t * const PublicKey_p, const
    SBIF_ECDSA_Signature_t * const Signature_p, uint32_t EcdsaWord) {
    _bootrom_jumtable_ecdsa_verify_init_t pTmp =
        (_bootrom_jumtable_ecdsa_verify_init_t)
(FUNC_ROM_ECDSA_VERIFY_INIT);
    return (*pTmp)(Verify_p, PublicKey_p, Signature_p, EcdsaWord);
}
```

/* points to bootrom_jumtable_ecdsa_verify_set_digest */

```
typedef void *
(*_bootrom_jumtable_ecdsa_verify_set_digest_t)
(SBHYBRID_EcdsaContext_t * const
    Verify_p, uint8_t * Digest_p);
inline static void *
adi_rom_EcdsaVerifySetDigest(SBHYBRID_EcdsaContext_t * const
    Verify_p, uint8_t * Digest_p) {
    _bootrom_jumtable_ecdsa_verify_set_digest_t pTmp =
```

```

        (_bootrom_jumtable_ecdsa_verify_set_digest_t)
(FUNC_ROM_ECDSA_VERIFY_SET_DIGEST);
    return (*pTmp)(Verify_p, Digest_p);
}

```

/* points to bootrom_jumtable_ecdsa_verify */

```

typedef SB_Result_t (*_bootrom_jumtable_ecdsa_verify_t)
(SBHYBRID_EcdsaContext_t *
    const Verify_p, uint32_t EcdsaWord);
inline static SB_Result_t adi_rom_EcdsaVerify(SBHYBRID_EcdsaContext_t *
const
    Verify_p, uint32_t EcdsaWord) {
    _bootrom_jumtable_ecdsa_verify_t pTmp =
        (_bootrom_jumtable_ecdsa_verify_t)(FUNC_ROM_ECDSA_VERIFY);
    return (*pTmp)(Verify_p, EcdsaWord);
}

```

Data Structures

The programming model for booting the processor uses the data structures defined in this section.

struct ADI_ROM_BOOT_BUFFER

Structure Type Declaration: ADI_ROM_BOOT_BUFFER

Boot Buffer.

A basic buffer type consisting of a pointer to the buffer and its size

Table 40-45: ADI_ROM_BOOT_BUFFER Members

Type	Name	Description
void *	pBuffer	Pointer to the buffer
int32_t	dByteCount	Size of the buffer

pBuffer

Pointer to the buffer

dByteCount

Size of the buffer

struct ADI_ROM_BOOT_CONFIG

Structure Type Declaration: ADI_ROM_BOOT_CONFIG

The Boot Configuration Object that contains all context for the boot process.

This structure contains the complete context for the boot process. A pointer to this object is passed through many routines and is presented to customizable routines such as initcodes, custom initialization, configuration, load and cleanup routines. The object is passed to error handlers and callbacks to customize and adapt the boot process for specific applications, especially in regards to multi-stage boot loader development.

Table 40-46: ADI_ROM_BOOT_CONFIG Members

Type	Name	Description
void *	pSource	Source address from where to fetch the next boot data.
void *	pDestination	Destination address to store the fetched data.
int32_t	dByteCount	Number of bytes to fetch from the boot source.
int32_t	dFlags	Control flags related to the boot kernel processing of blocks.
uint32_t	ulBlockCount	Limit of blocks processed during boot.
uint32_t	ulBlockCurrent	The number of blocks currently processed by the boot kernel
void *	pNextDxe	Pointer to the next application in the boot stream or the first free location after the boot stream.
uint32_t	uByteAddress	The destination address converted to the byte address space.
uint32_t volatile *	pControlRegister	Pointer to the boot peripherals control register.
int32_t	dControlValue	Storage for the boot peripheral main control value to enable that peripheral in a required configuration.
uint32_t volatile *	pPeripheralBase	Pointer to the boot peripherals base MMR address
uint32_t volatile *	pAuxControlRegister	Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection
uint32_t volatile *	pAuxPeripheralBase	Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block
uint32_t volatile *	pSecControlRegister	Base MMR address of the SEC SSI instance associated with the boot peripheral if required for advanced second stage boot loader development
ADI_DMA_TypeDef *	pDmaBaseRegister	Base MMR address of the DMA channel associated with the boot peripheral.
ROM_DMA_DONE_DETECT_METHOD	loadType	Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA
ROM_DMA_MDMA_CONFIG	MdmaCfg	An MDMA descriptor that is used by the boot kernel for internal MDMA operations.
uint16_t	uwDataWidth	The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit

Table 40-46: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
uint16_t	uwSrcModifyMult	The source modify multiplier used to set <code>DMA_XMOD</code> for source MDMA operations or peripheral DMA transmit operations
uint16_t	uwDstModifyMult	The destination modify multiplier used to set <code>DMA_XMOD</code> for destination MDMA operations or peripheral DMA receive operations
uint16_t	uwUserShort	Free to use
int32_t	dUserLong	Free to use
int32_t	dReserved0	Reserved for future use
void *	pModeData	Pointer to the boot mode specific data structure.
int32_t	dBootCommand	The boot command value supplied during the call to the <code>adi_rom_Boot()</code> routine.
ADI_ROM_BOOT_HEADER*	pHeader	Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel.
void *	pTempBuffer	Pointer to the internal intermediate buffer. Used for processing of indirect blocks.
void	dReserved1	Reserved
int32_t	dTempByteCount	Size of the internal intermediate buffer in bytes.
void *	pTempSource	Current source address that is processed in the internal intermediate buffer.
int32_t	dPageByteCount	The page size used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests.
ADI_ROM_BOOT_INTER_BUFFERS	bootBuffers	The internal intermediate buffer descriptors required when using indirect and page mode features.
ROM_BOOT_REGISTRY	bootRegistry	The registry object that is used to register a boot peripheral routines with the kernel.
ROM_BOOT_ERROR_FUNC *	pErrorFunction	Pointer to the error handler called in the event of an error.
ROM_BOOT_CALLBACK_FUNC *	pCallbackFunction	Pointer to the callback function that is called when processing boot blocks with the callback flag set.
ROM_BOOT_CALLBACK_FUNC *	pCrcFunction	Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data.
ROM_BOOT_CALLBACK_FUNC *	pForwardFunction	Feature not supported on this product.

Table 40-46: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
ADI_ROM_BOOT_MODES	bootModes	Access to all boot mode specific resources.
void *	pLogBuffer	Pointer to the log buffer. Logging is disabled by default on this product and must be configured from within initcodes or hook routines.
void *	pLogCurrent	The current position within the log buffer. Logging is disabled by default on this product and must be configured from within initcodes or hook routines.
int32_t	dLogByteCount	The size of the log buffer. Logging is disabled by default on this product and must be configured from within initcodes or hook routines.
ADI_ROM_OTP_BOOT_INFO*	pOtpBootInfo	Pointer to the ADI_ROM_OTP_BOOT_INFO boot information block that is read from OTP and contains boot customization options.
ADI_ROM_BOOT_KEY_TYPE	keyType	When set to a specific value allows keys not stored in OTP to perform secure boot evaluation on an open processor.
ADI_ROM_BOOT_TYPE	bootType	A key to indicate if the boot type is secure or non-secure for open parts.
ROM_SB_IMAGE_TYPE	secureBootImageType	The type of secure boot image.
SBIF_ECDSA_Header_t*	pSecureHeader	Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase.
ADI_SBIF_ECDSA_PublicKey_t	publicKey	The public key used for secure boot image authentication.
CRYPTO_DESCRIPTOR	cryptoDescriptors	The descriptor items as required for PKTE operations.
SB_StorageArea_t*	pSB_Storage	Storage area reserved for some crypto operations.
int32_t	secureBytesRemaining	The number of bytes remaining to be processed in the secure boot stream.
uint32_t[4]	aesKey	The 128-bit AES decryption key.
uint32_t[6]	aesWrapKey	The key wrapped key from the BLW secure boot image.
uint32_t[4]	IV	The IV as read from the secure boot header, required to initialize the PKTE.
uint8_t *	pHash	Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream.
uint32_t	ecdsaType	Type of ECDSA algorithm to sign the image

Table 40-46: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
uint32_t	errorReturn	Storage location for the address of the instruction line following a call to the error handler.

pSource

Source address from where to fetch the next boot data.

The source address must be maintained by the boot peripheral's load function. The kernel does not update the source pointer automatically after requesting data. This lets load routines control and change the source address. It is useful when advanced second stage loaders need to change the source address (due to a fragmented boot stream) or to reset the address (if expanding into a second SPI flash device).

During debug it is useful in identifying the block in the boot stream that is currently being processed.

pDestination

Destination address to store the fetched data.

Used by the boot kernel to indicate the destination address for the fetched data. A boot peripheral's load function must transfer the data to this location before returning back to the kernel. The boot kernel updates this field depending on whether a block header or payload is being fetched. In normal operation mode the kernel loads this field with the storage area location of the block header. After processing the block header loads `pDestination` with the `pTargetAddress` Destination address of payload contents read from the fetched block header. When using page mode the destination points to the internal buffers and is then updated to transfer data to the final destination.

dByteCount

Number of bytes to fetch from the boot source.

The kernel sets this parameter to indicate to the load function the number of bytes requested by the kernel. The kernel is responsible for adjusting the byte count for page mode based accesses. The peripheral load function must return the required number of bytes to the destination address provided.

dFlags

Control flags related to the boot kernel block processing.

When calling a boot mode using the `adi_rom_Boot()` routine the `flags` supplied to that routine are used to initialize `dFlags`. These become global flags that remain set through the entire boot process. When a block header is received the lower 16 bits of the block header are OR'ed with the global flags. The boot software may clear some flags if it detects they are not compatible with other flags and then writes the resulting flags back to this member. The boot kernel then processes the block payload as instructed by the combination of global and boot block specific flags. Upon completion of the block processing original set of global flags are restored and the process repeated.

ulBlockCount

Limit of blocks processed during boot.

When calling the boot process the `adi_rom_Boot()` routine can accept a block limit for the number of blocks to process before terminating the boot process. If the block count is set to zero then the boot process will continue until a final block reached indicating end of the boot stream. This member holds the program's specified limit for the number of blocks to process and is used by the boot kernel after processing each block to compare it against the `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` value. Boot process terminates when `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` equals `ADI_ROM_BOOT_CONFIG::ulBlockCount`

ulBlockCurrent

The number of blocks currently processed by the boot kernel

pNextDxe

Pointer to the next application in the boot stream or the first free location after the boot stream.

This member is initialized when processing a first block in the boot stream. The `dArgument` `Argument` function varies depending on operation field of a first block contains the number of bytes left in the boot stream before we reach the end of that boot stream. This lets this pointer point to the next boot stream or to the first empty location after the boot stream. This allows for a feature when using the `adi_rom_Boot()` routine to find the address of an application in a linked list of boot streams or to find the first empty location after the boot stream.

uByteAddress

The destination address converted to the byte address space.

This member is used to store the byte address space equivalent of SHARC L1 memory addresses, allowing any core to load content to any SHARC cores L1 memory. The SHARC core in which the load is targeted is determined by the block header.

pControlRegister

Pointer to the boot peripherals control register.

This can be used by a boot mode peripherals driver in order to gain efficient access to a control MMR register in the boot peripheral.

NOTE: This is not used in this products boot implementation but may be leveraged by developers of second stage boot loaders if required

dControlValue

Storage for the boot peripheral main control value to enable that peripheral in a required configuration.

This can be used by a boot modes peripheral driver in order to store a control value that can be used to enable the peripheral for a required configuration.

pPeripheralBase

Pointer to the boot peripherals base MMR address

pAuxControlRegister

Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection

pAuxPeripheralBase

Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block

pSecControlRegister

Base MMR address of the SEC SSI instance associated with the boot peripheral should they be required for advanced second stage boot loader development

pDmaBaseRegister

Base MMR address of the DMA channel associated with the boot peripheral.

This is used by the boot kernel to gain access to the DMA channels status when using non blocking DMA operations when page mode or secure boot is required, so it must be set when implementing custom boot loaders in order for the kernel to get access to that peripherals DMA status.

NOTE: If a custom boot peripheral does not support the standard DMA instance then the custom driver will be required to set up a DMA instance in SRAM that this location points to and the load function would need to update the status accordingly to indicate when the DMA was running and when the DMA completed.

loadType

Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA

MdmaCfg

An MDMA descriptor that is used by the boot kernel for internal MDMA operations.

The boot kernel may be required to perform internal MDMA operations outside the control of the boot peripheral driver. Such operations include processing of fill blocks CRC callbacks for CRC verification and MDMA operations from the internal intermediate buffers for indirect block and page mode processing.

NOTE: Users must not use this item or reconfigure this item when developing custom boot drivers, it is intended purely for the internal use by the boot kernel

uwDataWidth

The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit

uwSrcModifyMult

The source modify multiplier used to set `DMA_XMOD` for source MDMA operations or peripheral DMA transmit operations

uwDstModifyMult

The destination modify multiplier used to set `DMA_XMOD` for destination MDMA operations or peripheral DMA receive operations

uwUserShort

Free to use by the user

dUserLong

Free to use by the user

pModeData

Pointer to the boot mode specific data structure.

Can be set by a boot peripheral driver to allow for a single point of access to the boot mode specific object containing control and configuration information specific to that single boot mode.

dBootCommand

The boot command value supplied during the call to the `adi_rom_Boot()` routine

pHeader

Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel

pTempBuffer

Pointer to the internal intermediate buffer. Used for processing of indirect blocks

dTempByteCount

Size of the internal intermediate buffer in bytes

pTempSource

Current source address that is being processed in the internal intermediate buffer

dPageByteCount

The page size used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests

bootBuffers

The internal intermediate buffer descriptors required when using indirect and page mode features

bootRegistry

The registry object that is used to register a boot peripherals routines with the kernel.

When using the `adi_rom_Boot()` function the boot software calls a peripherals initialization, configuration, load and cleanup routines from the pointers stored in this object. The kernel itself only makes calls to the load function for the peripheral so when using the `adi_rom_BootKernel()` function the load function that is called by the boot kernel to fetch data from the boot source must be registered using `pLoadFunctionPointer` to the boot modes Load function .

pErrorFunction

Pointer to the error handler called in the event of an error

pCallbackFunction

Pointer to the callback function that is called when processing boot blocks with the callback flag set

pCrcFunction

Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data

pForwardFunction

Feature not supported on this product

bootModes

Access to all boot mode specific resources

pLogBuffer

Pointer to the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pLogCurrent

The current position within the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

dLogByteCount

The size of the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pOtpBootInfo

Pointer to the `struct ADI_ROM_OTP_BOOT_INFO` boot information block that gets read from OTP and contains boot customization options

keyType

When set to a specific value allows keys not stored in OTP to be used for secure boot evaluation on an open processor.

By default when performing boot on an open part the decryption keys and the public key are fetched from OTP. By setting this field to `enum ADI_ROM_BOOT_KEY_TYPE` users can disable the fetching of the keys from OTP and instead provision the keys directly in the `ADI_ROM_BOOT_CONFIG::publicKey` and `ADI_ROM_BOOT_CONFIG::aesKey` members via hook routines when using the `adi_rom_Boot()` function.

bootType

A key to indicate if the boot type is secure or non-secure for open parts

secureBootImageType

The type of secure boot image

pSecureHeader

Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase

publicKey

The public key used for secure boot image authentication

cryptoDescriptors

The descriptor items as required for the PKTE operations

pSB_Storage

Storage area reserved for some crypto operations

secureBytesRemaining

The number of bytes remaining to be processed in the secure boot stream

aesKey

The 128-bit AES decryption key

aesWrapKey

The key wrapped key from the BLW secure boot image

IV

The IV as read from the secure boot header as required to initialize the PKTE

pHash

Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream

ecdsaType

This holds the ecdsa type to be used for authentication. It can be 224-bit or 256-bit.

errorReturn

Storage location for the address of the instruction line following a call to the error handler

struct ADI_ROM_BOOT_HEADER

Structure Type Declaration: ADI_ROM_BOOT_HEADER

Boot Block Header.

Boot block headers control the loading process of the boot stream, For full details on the contents of the block header and supported flags see [Boot Loader Stream](#).

Table 40-47: ADI_ROM_BOOT_HEADER Members

Type	Name	Description
int32_t	dBlockCode	Instructs the boot kernel how to process the block.
void *	pTargetAddress	Destination Address of Payload
int32_t	dByteCount	Byte Count of the Payload
int32_t	dArgument	Argument functionality varies depending on operation

dBlockCode

Instructs the boot kernel how to process the block.

Contains a number of fields for verification of the block header and flags to indicate the type of block. This allows the kernel to process the block correctly.

pTargetAddress

Destination address of payload

dByteCount

Byte count of the payload

dArgument

Argument function varies depending on operation

struct ADI_ROM_BOOT_INTER_BUFFER

Structure Type Declaration: ADI_ROM_BOOT_INTER_BUFFER

The buffer object for the internal intermediate buffers used for indirect and page mode operations.

Table 40-48: ADI_ROM_BOOT_INTER_BUFFER Members

Type	Name	Description
uint8_t *	pBuffer	Pointer to the buffer
uint32_t	size	Size of the buffer
uint32_t	pageSize	Page size for block based devices.

pBuffer

Pointer to the buffer

size

Size of the buffer

pageSize

Page size for block based devices.

NOTE: This field is not used in this product. A fixed page size of 1024 bytes is used.

struct ADI_ROM_BOOT_INTER_BUFFERS

Structure Type Declaration: ADI_ROM_BOOT_INTER_BUFFERS

The boot kernels internal buffer object used to access the intermediate buffers and obtain buffer status.

Table 40-49: ADI_ROM_BOOT_INTER_BUFFERS Members

Type	Name	Description
ADI_ROM_BOOT_INTER_BUFFER[2]	buffer	The two buffer descriptors
ADI_ROM_BOOT_BUFFER_STATE	state	Buffer Status Information
void *	pSource	Original source address pointer of data loaded to active buffer. Not used on this product
ADI_DMA_TypeDef *	pDma	TBD

buffer

The two buffer descriptors

state

Buffer Status Information

pSource

Original source address pointer of data loaded to active buffer. Not used on this product

struct ADI_ROM_BOOT_LINKPORT

Structure Type Declaration: ADI_ROM_BOOT_LINKPORT

The linkport slave boot mode specific structure.

This structure contains all the boot context information that is specific to the linkport slave boot mode.

Table 40-50: ADI_ROM_BOOT_LINKPORT Members

Type	Name	Description
uint32_t	nFlags	Flags related to linkport boot mode
ADI_LP_TypeDef *	pRegisters	Pointer to the LP peripherals base MMR address, not used on this product
ADI_DMA_TypeDef *	pRxDmaRegisters	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
ADI_DMA_TypeDef *	pTxDmaRegisters	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

nFlags

Flags related to linkport boot mode

pRegisters

Pointer to the LP peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

struct ADI_ROM_BOOT_MODES

Structure Type Declaration: ADI_ROM_BOOT_MODES

Holds all boot mode specific configuration items.

A boot mode may have requirements for some dedicated storage. This object is used to collect all storage items for all the boot modes supported by the boot ROM.

Table 40-51: ADI_ROM_BOOT_MODES Members

Type	Name	Description
ADI_ROM_BOOT_SPI	spi	Access to all SPI boot mode resources
ADI_ROM_BOOT_UART	uart	Access to all UART boot mode resources

Table 40-51: ADI_ROM_BOOT_MODES Members (Continued)

Type	Name	Description
ADI_ROM_BOOT_LINKPORT	linkport	Access to all LINKPORT boot mode resources
ADI_ROM_BOOT_OSPI	spi3	Access to all OSPI boot mode resources
ADI_ROM_BOOT_CUSTOM	custom	Access to all custom boot mode resources

spi

Access to all SPI boot mode resources

uart

Access to all UART boot mode resources

linkport

Access to all LINKPORT boot mode resources

custom

Access to all custom boot mode resources

Spi3

Access to all OSPI boot mode resources

struct ADI_ROM_BOOT_REGISTRY

Structure Type Declaration: ADI_ROM_BOOT_REGISTRY

Boot Mode Registration.

Used to hold pointers for the boot mode initialization, configuration, load and cleanup functions. Can customize the registered content via hook routines or install load functions or cleanup functions from within init codes.

When using `adi_rom_Boot()` the boot process makes a call to the initialization function and the configuration function before calling the kernel. The kernel then runs and makes calls to the load function. The cleanup function is called when the kernel reaches the end of the boot stream.

When using the `adi_rom_BootKernel()` function only the load function is called during execution of the software in the boot ROM. All the functions here must return a `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` result in order for the boot process to continue. All functions expect a single argument that is the pointer to the boot structure object `struct ADI_ROM_BOOT_CONFIG`.

Table 40-52: ADI_ROM_BOOT_REGISTRY Members

Type	Name	Description
ROM_BOOT_MODE_INIT_FUNC *	pInitFunction	Pointer to the boot modes Initialization function
ROM_BOOT_MODE_CONFIG_FUNC *	pConfigFunction	Pointer to the boot modes Configuration function
ROM_BOOT_MODE_LOAD_FUNC *	pLoadFunction	Pointer to the boot modes Load function
ROM_BOOT_MODE_CLEANUP_FUNC *	pCleanUpFunction	Pointer to the boot modes Cleanup function
void *	pReserved	Reserved for future use
int32_t	dReserved	Reserved for future use

pInitFunction

Pointer to the boot modes Initialization function

pConfigFunction

Pointer to the boot modes Configuration function

pLoadFunction

Pointer to the boot modes Load function

pCleanUpFunction

Pointer to the boot modes Cleanup function

struct ADI_ROM_BOOT_SPI

Structure Type Declaration: ADI_ROM_BOOT_SPI

The SPI Master Boot Mode Specific Structure.

This structure contains all the boot context information that is specific for SPI master boot mode. During auto-detection information is copied from the required ::ROM_SPI_LUTENTRY item into this structure and used to configure the SPI peripheral for the mode of operation.

Table 40-53: ADI_ROM_BOOT_SPI Members

Type	Name	Description
uint8_t	ubReadCommand	Read command to read data from the SPI device
uint8_t	ubDummyBytes	Number of dummy bytes to issue after the read command
uint8_t	ubAddressBytes	Number of address bytes required to access the device
uint8_t	ubDataBits	Bus width when reading the data. 0 = single bit, 1 = dual bit, 2 = quad bit

Table 40-53: ADI_ROM_BOOT_SPI Members (Continued)

Type	Name	Description
uint16_t	uwClkLower	SPI clock divider value
uint16_t	uReserved0	Reserved
uint32_t	nTxCtl	The value written to the <code>SPI_TXCTL</code> register used for address transmit operations such as address cycles
uint32_t	nRxCtl	The value written to the <code>SPI_RXCTL</code> register that is used for all receive operations
uint32_t	nCmdCtl	The value written to the <code>SPI_TXCTL</code> register that is used for sending the read command to the SPI Flash
ROM_BOOT_SPIM_IO_ENABLE_FUNC *	pMIOEnFunction	Pointer to the function used to enable quad mode on the SPI flash
uint8_t	nDummy	The Dummy Byte value if dummy byte transfers are required and the bus is not three-stated
uint8_t	nFlags	Flags used for additional SPI configuration processing.
uint16_t	uReserved2	Reserved
void *	pXIPAddress	The memory-mapped SPI address to boot from
ADI_SPI_TypeDef *	pRegisters	Pointer to the SPI peripherals base MMR address, not used on this product
ADI_DMA_TypeDef *	pRxDmaRegisters	Pointer to the DMA peripherals base MMR address, for receive operations, not used on this product
ADI_DMA_TypeDef *	pTxDmaRegisters	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

ubReadCommand

Read command to use to read data from the SPI device

ubDummyBytes

Number of dummy bytes to issue after the read command

ubAddressBytes

Number of address bytes required to access the device

ubDataBits

The bus width used when reading the data. 0 for single bit, 1 for dual, 2 for quad

uwClkLower

The SPI clock divider value

nTxCtl

The value written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles

nRxCtl

The value written to the `SPI_RXCTL` register that is used for all receive operations

nCmdCtl

The value written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash

pMIOEnFunction

Pointer to the function used to enable quad mode on the SPI flash

nDummy

The Dummy Byte value if dummy byte transfers are required and the bus is not tri-stated

nFlags

Flags used for some additional SPI configuration processing. The flags supported are defined as follows:

Bit Position	Name	Description
0	<code>ROM_SPI_FLAGS_CMDSKIP_EN</code>	When set the configuration routine enables command skip mode where the SPI does not issue a read command for read operations
1	<code>ROM_SPI_FLAGS_MULTICMD_EN</code>	Configuration routine enables sending command cycles over dual or quad bit bus

pXIPAddress

The memory-mapped SPI address to boot from

pRegisters

Pointer to the SPI peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address, for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address, for transmit operations, not used on this product

struct ADI_ROM_BOOT_OSPI

Structure Type Declaration: `ADI_ROM_BOOT_OSPI`

The SPI Master Boot Mode Specific Structure.

This structure contains all the boot context information that is specific for OSPI master boot mode. During auto-detection information is copied from the required `ROM_OSPI_LUTENTRY` item into this structure and used to configure the OSPI peripheral for the mode of operation.

Table 40-54: ADI_ROM_BOOT_OSPI Members

Type	Name	Description
uint8_t	ubReadCommand	Read command to read data from the OSPI device
uint8_t	ubDummyCycles	Number of dummy cycles to issue after the read command
uint8_t	ubAddressBytes	Number of address bytes required to access the device
uint8_t	ubDataBits	Bus width when reading the data
uint16_t	uwClkLower	OSPI BAUD rate divisor in the <code>OSPI_CTL.BAUD</code> register
uint16_t	uReserved0	Reserved
uint32_t	nCfg	Value written to Lower 16 bits of the <code>OSPI_CTL</code> register
uint32_t	nDsr	OSPI Device Size written to the <code>OSPI_DSCTL</code> register
uint32_t	nDrir	OSPI Device Read Instruction Control value written to the <code>OSPI_DRICTL</code> register
uint32_t	reserved1	Reserved
uint8_t	nDummy	The dummy byte value to use
uint8_t	nFlags	Flags for additional SPI configuration processing
uint16_t	uReserved2	Reserved
void *	pXIPAddress	The memory-mapped OSPI address to boot
ADI_OSPI_TypeDef *	pRegisters	Reserved
ADI_DMA_TypeDef *	pRxDmaRegisters	Reserved
ADI_DMA_TypeDef *	pTxDmaRegisters	Reserved

ubReadCommand

Read command to use to read data from the SPI device

ubDummyCycles

Number of dummy cycles to issue after the read command

ubAddressBytes

Number of address bytes required to access the device

ubDataBits

The bus width used when reading the data. 0 for single bit, 1 for dual, 2 for quad

uwClkLower

The SPI clock divider value

nCfg

The value written to the Lower 16 bits of the OSPI Control register

nDsr

The value written to the OSPI Device Size register

nDrir

The value written to the OSPI Device Read Instruction Control register

nDummy

The Dummy Byte value if dummy byte transfers are required and the bus is not tri-stated

nFlags

Flags used for some additional OSPI configuration processing. The flags supported are defined as follows:

- `ROM_SPI_FLAGS_CMDSKIP_EN` (Bit position 0): When set the configuration routine enables command skip mode where the SPI does not issue a read command for read operations
- `ROM_SPI_FLAGS_MULTICMD_EN` (Bit position 1): Configuration routine enables sending command cycles over dual or quad bit bus

pXIPAddress

The memory-mapped SPI address to boot

pRegisters

Pointer to the OSPI peripherals base MMR address (not used on this product)

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address for receive operations (not used on this product)

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address, for transmit operations (not used on this product)

struct ADI_ROM_BOOT_UART

Structure Type Declaration: `ADI_ROM_BOOT_UART`

The UART slave boot mode specific structure.

This structure contains all the boot context information that is specific for the UART slave boot mode.

Table 40-55: ADI_ROM_BOOT_UART Members

Type	Name	Description
uint32_t	nFlags	Flags related to UART Boot mode
ADI_UART_TypeDef*	pRegisters	Pointer to the UART peripherals base MMR address, not used on this product
ADI_DMA_TypeDef*	pRxDmaRegisters	Pointer to the DMA peripherals base MMR address for receive operations, not used on this product
ADI_DMA_TypeDef*	pTxDmaRegisters	Pointer to the DMA peripherals base MMR address for transmit operations, not used on this product

nFlags

Flags related to UART Boot mode

pRegisters

Pointer to the UART peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address for transmit operations, not used on this product

struct ADI_ROM_OTP_BOOT_CFG

Structure Type Declaration: ADI_ROM_OTP_BOOT_CFG

The boot configuration object for storing further boot customization objects.

This is a 160-bit structure that is allocated to one contiguous region in the OTP memory array. The functionality allows for individual flags to enable or disable specific features of the boot process. Each flag is allocated in a separate 16-bit word so that each flag can be set at different times and the ECC information will not impact the setting of another flag.

Table 40-56: ADI_ROM_OTP_BOOT_CFG Members

Type	Name	Description
uint32_t	lockMonitor:1 (bitfield)	Flag to enable the status checking of ADI private OTP area
uint32_t	reserved0:31 (bitfield)	Reserved
uint32_t	pubkey0Inv:1 (bitfield)	Invalidate Public Key 0, use next public key for secure boot

Table 40-56: ADI_ROM_OTP_BOOT_CFG Members (Continued)

Type	Name	Description
uint32_t	reserved1:15 (bitfield)	Reserved
uint32_t	pubkey1Inv:1 (bitfield)	Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys
uint32_t	reserved2:15 (bitfield)	Reserved
uint32_t	privkey0Inv:1 (bitfield)	Invalidate Decryption Key 0, use next Decryption key for secure boot
uint32_t	reserved3:15 (bitfield)	Reserved
uint32_t	privkey1Inv:1 (bitfield)	Invalidate Decryption Key 1, use next Decryption key for secure boot
uint32_t	reserved4:15 (bitfield)	Reserved
uint32_t	privkey2Inv:1 (bitfield)	Invalidate Decryption Key 2, use next Decryption key for secure boot
uint32_t	reserved5:15 (bitfield)	Reserved
uint32_t	privkey3Inv:1 (bitfield)	Invalidate Decryption Key 3, once invalidated part is no longer bootable
uint32_t	reserved6:15 (bitfield)	Reserved
uint32_t	dmcEn:1 (bitfield)	Enables configuration of the DMC from values in OTP stored in the format of the ADI_ROM_OTP_DMC_CONFIG object
uint32_t	reserved7:15 (bitfield)	Reserved
uint32_t	dmcInv:1 (bitfield)	Invalidates the DMC values in the OTP resulting is bypassing of DMC configuration
uint32_t	reserved8:15 (bitfield)	Reserved

lockMonitor

Flag to enable the status checking of ADI private OTP area.

pubkey0Inv

Invalidate Public Key 0, use next public key for secure boot

pubkey1Inv

Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys

privkey0Inv

Invalidate Decryption Key 0, use next Decryption key for secure boot

privkey1Inv

Invalidate Decryption Key 1, use next Decryption key for secure boot

privkey2Inv

Invalidate Decryption Key 2, use next Decryption key for secure boot

privkey3Inv

Invalidate Decryption Key 3, once invalidated part will no longer be bootable

dmcEn

Enables configuration of the DMC from values in OTP stored in the format of the `ADI_ROM_OTP_DMC_CONFIG` object

dmcInv

Invalidates the DMC values in the OTP resulting in bypassing of DMC configuration

struct ADI_ROM_OTP_BOOT_CGU_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_CGU_INFO`

The CGU configuration object located in OTP for configuration of the CGU by the boot software.

This is a 96-bit structure that is allocated to one contiguous region in the OTP memory array. Allows the boot software to configure the CGU for a more efficient boot process.

Table 40-57: `ADI_ROM_OTP_BOOT_CGU_INFO` Members

Type	Name	Description
<code>uint32_t</code>	<code>ctl_WEN:1</code> (bitfield)	Enable write to the <code>CGU_CTL</code> register
<code>uint32_t</code>	<code>div_WEN:1</code> (bitfield)	Enable write to the <code>CGU_DIV</code> register
<code>uint32_t</code>	<code>reserved0:1</code> (bitfield)	Reserved
<code>uint32_t</code>	<code>div_DSEL:5</code> (bitfield)	<code>CGU_DIV.DSEL</code> value
<code>uint32_t</code>	<code>div_CSEL:5</code> (bitfield)	<code>CGU_DIV.CSEL</code> value
<code>uint32_t</code>	<code>div_SOSEL:3</code> (bitfield)	<code>CGU_DIV.SOSEL</code> value
<code>uint32_t</code>	<code>div_SYSEL:5</code> (bitfield)	<code>CGU_DIV.SYSEL</code> value
<code>uint32_t</code>	<code>div_S1SEL:3</code> (bitfield)	<code>CGU_DIV.S1SEL</code> value
<code>uint32_t</code>	<code>div_OSEL:7</code> (bitfield)	<code>CGU_DIV.OSEL</code> value
<code>uint32_t</code>	<code>ctl_DF:1</code> (bitfield)	<code>CGU_CTL.DF</code> value

Table 40-57: ADI_ROM_OTP_BOOT_CGU_INFO Members (Continued)

Type	Name	Description
uint32_t	ctl_MSEL:7 (bitfield)	CGU_CTL.MSEL value
uint32_t	auto_disable:1 (bitfield)	disable polling on auto-alignment of clocks (not recommended)
uint32_t	Reserved1:6 (bitfield)	Reserved
uint32_t	clkoutsel_CLKOUTSEL:5 (bitfield)	CGU_CLKOUTSEL.CLKOUTSEL value
uint32_t	clkoutsel_WEN:1 (bitfield)	Enable write to the CGU_CLKOUTSEL register
uint32_t	Reserved2:12 (bitfield)	Reserved
uint32_t	oscwctl0_WEN:1 (bitfield)	Enable write to the CGU_OSCWDCTL instance 0 register
uint32_t	oscwctl0_HODF:6 (bitfield)	CGU_OSCWDCTL.HODF value
uint32_t	oscwctl0_HODEN:1 (bitfield)	CGU_OSCWDCTL.HODEN value
uint32_t	oscwctl0_CNGEN:1 (bitfield)	CGU_OSCWDCTL.CNGEN value
uint32_t	oscwctl0_BOUF:5 (bitfield)	CGU_OSCWDCTL.BOUF value
uint32_t	oscwctl0_BOUEN:1 (bitfield)	CGU_OSCWDCTL.BOUEN value
uint32_t	oscwctl0_FAULTEN:1 (bitfield)	CGU_OSCWDCTL.FAULTEN value
uint32_t	oscwctl0_MONDIS:1 (bitfield)	CGU_OSCWDCTL.MONDIS value
uint32_t	oscwctl0_FAULTPINDIS:1 (bitfield)	CGU_OSCWDCTL.FAULTPINDIS value
uint32_t	Reserved3:14 (bitfield)	Reserved

ctl_WEN

Enable write to the [CGU_CTL](#) register

div_WEN

Enable write to the [CGU_DIV](#) register

div_DSEL

CGU_DIV.DSEL value

div_CSEL

CGU_DIV.CSEL value

div_S0SEL

CGU_DIV.S0SEL value

div_SYSSSEL

CGU_DIV.SYSSSEL value

div_S1SEL

CGU_DIV.S1SEL value

div_OSEL

CGU_DIV.OSEL value

ctl_DF

CGU_CTL.DF value

ctl_MSEL

CGU_CTL.MSEL value

auto_disable

disable polling on auto-alignment of clocks, NOT RECOMMENDED!

clkoutsel_CLKOUTSEL

CGU_CLKOUTSEL.CLKOUTSEL value

clkoutsel_WENEnable write to the [CGU_CLKOUTSEL](#) register**oswctl0_WEN**Enable write to the [CGU_OSCWDCTL](#) instance 0 register**oswctl0_HODF**

CGU_OSCWDCTL.HODF value

oswctl0_HODEN

CGU_OSCWDCTL.HODEN value

oswctl0_CNGEN

CGU_OSCWDCTL.CNGEN value

oswctl0_BOUF

CGU_OSCWDCTL.BOUF value

oswctl0_BOUEN

CGU_OSCWDCTL.BOUEN value

oswctl0_FAULTEN

CGU_OSCWDCTL.FAULTEN value

oswctl0_MONDIS

`CGU_OSCWDCTL.MONDIS` value

`oswctl0_FAULTPINDIS`

`CGU_OSCWDCTL.FAULTPINDIS` value

struct ADI_ROM_OTP_BOOT_CMD_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_CMD_INFO`

The boot command object for storing a customized boot command for each boot mode within the OTP memory array.

This is a 160-bit structure that is allocated to one contiguous region in the OTP memory array. Allows the boot ROM software to pass a custom boot command to a specific boot mode which changes the default boot behavior on startup. Can be used to change the default UART instance used for a UART boot operation (for example).

Table 40-58: `ADI_ROM_OTP_BOOT_CMD_INFO` Members

Type	Name	Description
<code>uint32_t</code>	<code>spiMasterBootCmd</code>	SPI Master Boot Mode
<code>uint32_t</code>	<code>spiSlaveBootCmd</code>	SPI Slave Boot Mode
<code>uint32_t</code>	<code>lpBootCmd</code>	Link Port Slave Boot Mode
<code>uint32_t</code>	<code>uartBootCmd</code>	UART Slave Boot Mode
<code>uint32_t</code>	<code>ospiMasterBootCmd</code>	OSPI Master Boot Mode

`spiMasterBootCmd`

[SPI Master Boot Mode](#)

`spiSlaveBootCmd`

[SPI Slave Boot Mode](#)

`lpBootCmd`

[Link Port Slave Boot Mode](#)

`uartBootCmd`

[UART Slave Boot Mode](#)

`ospiMasterBootCmd`

[OSPI Master Boot Mode](#)

struct ADI_ROM_OTP_BOOT_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_INFO`

The 544-bit boot info object located in OTP for boot customization.

This is a 544 bit structure that is allocated to one contiguous region in the OTP memory array. The content in OTP is stored in the format of this structure so boot can read the contents directly into this object.

Programs can read this object using the `adi_rom_otp_get()` routine supplying the `enum OTPCMD` enumeration

Table 40-59: ADI_ROM_OTP_BOOT_INFO Members

Type	Name	Description
<code>struct ADI_ROM_OTP_BOOT_CGU_INFO</code>	<code>cgu</code>	CGU Configuration information
<code>uint32_t</code>	<code>flashStartAddress</code>	Flash Start address for the SPI/OSPI master boot mode
<code>struct ADI_ROM_OTP_BOOT_CMD_INFO</code>	<code>bcmd</code>	Boot Command customization for each boot mode to change default boot peripheral instance and configuration
<code>struct ADI_ROM_OTP_BOOT_CFG</code>	<code>bcfg</code>	Additional boot configuration flags for key invalidation and to enable DMC configuration
<code>uint32_t</code>	<code>reserved0</code>	Reserved
<code>uint16_t</code>	<code>otpReadTiming</code>	Reserved, Must always be zero
<code>uint16_t</code>	<code>reserved1</code>	Reserved

cgu

CGU Configuration information

flashStartAddress

Flash start address for the SPI/OSPI master boot mode

bcmd

Boot Command customization for each boot mode to change the default boot peripheral instance and configuration

bcfg

Additional boot configuration flags for key invalidation and for enabling DMC configuration

otpReadTiming

Reserved, Must always be zero

struct ADI_ROM_OTP_DMC_CONFIG

Structure Type Declaration: `ADI_ROM_OTP_DMC_CONFIG`

The 384-bit configuration object located in OTP for configuration of the DMC during preboot.

Makes use of the memories connected to the DMC peripheral during boot without the use of init codes in the boot stream. The settings can be applied to this object in the OTP memory. During preboot the boot software reads the object from OTP and configures the peripheral accordingly.

NOTE: When the device is open programs should avoid using OTP and instead use initcodes to initialize DMC because the initcode method is highly customizable. To lock the device to enable secure boot (to boot to memories interfaced to the DMC) the configuration must be provisioned to OTP because initcodes are not supported in secure boot.

Table 40-60: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
uint32_t	ulDDR_DLLCTLCFG	Content of <code>DMC_DLLCTL</code> and <code>DMC_CFG</code> registers
uint32_t	ulDDR_EMR2EMR3	Content of the <code>DMC_MR2</code> and <code>DMC_EMR3</code> registers
uint32_t	ulDDR_CTL	Content of the <code>DMC_CTL</code> register
uint32_t	ulDDR_MREMR1	Content of the <code>DMC_MR</code> and <code>DMC_MR1</code> register
uint32_t	ulDDR_TR0	Content of the <code>DMC_TR0</code> register
uint32_t	ulDDR_TR1	Content of the <code>DMC_TR1</code> register
uint32_t	ulDDR_TR2	Content of the <code>DMC_TR2</code> register
uint32_t	ulDDR_ZQCTL0	Content of <code>DMC_DDR_ZQ_CTL0</code> register
uint32_t	ulDDR_ZQCTL1	Content of <code>DMC_DDR_ZQ_CTL1</code> register
uint32_t	ulDDR_ZQCTL2	Content of <code>DMC_DDR_ZQ_CTL2</code> register
uint32_t	ulDDRRPHY_CACTL	Content of <code>DMC_DDR_CA_CTL</code> register
uint32_t	uBypassDelay_LANE0CTL1:6	Content of <code>DMC_DDR_LANE0_CTL1[15:10]</code> register
uint32_t	uBypassDelay_LANE1CTL1:6	Content of <code>DMC_DDR_LANE1_CTL1[15:10]</code> register
uint32_t	uBypassDelay_LANE0CTL0:6	Content of <code>DMC_DDR_LANE0_CTL0[15:10]</code> register
uint32_t	uBypassDelay_LANE1CTL0:6	Content of <code>DMC_DDR_LANE1_CTL0[15:10]</code> register
uint32_t	reserved0:8	Reserved

struct ROM_BOOT_DMA_INSTANCE

Structure Type Declaration: `ROM_BOOT_DMA_INSTANCE`

DMA Channel Instance.

Specifies the base MMR address of the DMA channel as well as trigger and interrupt IDs

Table 40-61: ROM_BOOT_DMA_INSTANCE Members

Type	Name	Description
ADI_DMA_TypeDef *	pReg	Pointer to the base address of the DMA channel MMR registers
DMA_CHANn_TypeDef	eDmaChannelId	The actual DMA channel ID in the system
uint8_t	TriggerId	The trigger ID associated with the DMA channel
uint8_t	InterruptId	The interrupt ID associated with the DMA channel

pReg

Pointer to the base address of the DMA channel MMR registers

eDmaChannelId

The actual DMA channel ID in the system

TriggerId

The trigger ID associated with the DMA channel

InterruptId

The interrupt ID associated with the DMA channel

struct ROM_BOOT_MDMA

Structure Type Declaration: ROM_BOOT_MDMA

MDMA Channels.

Provides access to all the MDMA channels and CRC peripherals supported by the processor

Table 40-62: ROM_BOOT_MDMA Members

Type	Name	Description
ROM_BOOT_MDMA_REGS [PARAM_ SYS0_NUM_MDMA_STREAMS]	Stream	Array of MDMA channel configurations supported by the processor

Stream

Array of MDMA channel configurations supported by the processor

struct ROM_BOOT_MDMA_REGS

Structure Type Declaration: ROM_BOOT_MDMA_REGS

MDMA Channel Registers.

Contains the Source and Destination MDMA channel instances for access to the MMRs and interrupt and trigger information. Information is also provided on the CRC support of the MDMA channel and access is provided to the corresponding CRC peripheral.

Table 40-63: ROM_BOOT_MDMA_REGS Members

Type	Name	Description
struct ROM_BOOT_DMA_INSTANCE	Src	The source DMA Channel in the MDMA pair
ROM_BOOT_DMA_INSTANCE	Dst	The destination DMA Channel in the MDMA pair
ADI_CRC_TypeDef *	pCrc	The base MMR address of the associated CRC peripheral if one exists
enum ROM_BOOT_MDMA_CRC_SUPPORT	eCrcSupport	Indicates if the MDMA channel supports CRC or not

Src

The source DMA Channel in the MDMA pair

Dst

The destination DMA Channel in the MDMA pair

pCrc

The base MMR address of the associated CRC peripheral if one exists

eCrcSupport

Indicates if the MDMA channel supports CRC or not

struct ROM_DMA_MDMA_CONFIG

Structure Type Declaration: ROM_DMA_MDMA_CONFIG

MDMA Configuration Object.

The configurable structure for controlling the MDMA operation supplied to the [adi_rom_MemDma\(\)](#) routine.

Table 40-64: ROM_DMA_MDMA_CONFIG Members

Type	Name	Description
enum ROM_DMA_MDMA_OPERATION	eOperation	Type of operation to perform
enum ROM_DMA_MDMA_ID	eId	MDMA Channel ID
void *	pSource	Source Pointer
void *	pDestination	Destination Pointer

Table 40-64: ROM_DMA_MDMA_CONFIG Members (Continued)

Type	Name	Description
uint32_t	ByteCount	Byte Count
enum ROM_DMA_DONE_DETECT_METHOD	eDoneDetect	DMA Done Detection Method
uint32_t	CrcCtl	CRC_CTL value when CRC operations are required
uint32_t	FillVal	Fill value for memory fill operations
uint32_t	CrcPoly	CRC Polynomial for CRC operations
uint32_t	CrcCompare	Value used for CRC compare operations or for a CRC32 result compare

eOperation

Type of operation to perform

eId

MDMA Channel ID

pSource

Source Pointer

pDestination

Destination Pointer

ByteCount

Byte Count

eDoneDetect

DMA Done Detection Method

CrcCtl

CRC_CTL value when CRC operations are required

FillVal

Fill value for memory fill operations

CrcPoly

CRC Polynomial for CRC operations

CrcCompare

Value used for CRC compare operations or for a CRC32 result compare

struct ROM_DMA_PDMA_CONFIG

Structure Type Declaration: ROM_DMA_PDMA_CONFIG

PDMA Configuration Object.

The user configurable structure for controlling the PDMA operation via the [adi_rom_PeriphDma\(\)](#) function.

Table 40-65: ROM_DMA_PDMA_CONFIG Members

Type	Name	Description
enum ROM_DMA_PDMA_OPERATION	eOperation	Type of operation to perform
ADI_DMA_TypeDef volatile *	pRegs	Pointer to the base address of the DMA channel MMR registers
uint16_t	dataWidth	The maximum supported data width of the DMA channel. Used to configure the DMA_CFG.PSIZE PSIZE field in DMA_CFG
uint16_t	dstModifyMult	The modify multiplier, usually set to 1
void *	pSource	Source Pointer used for transmit operations
void *	pDestination	Destination Pointer used for receive operations
uint32_t	byteCount	Number of bytes to transfer
enum ROM_DMA_DONE_DETECT_METHOD	eDoneDetect	DMA Done Detection method used for the transfer
enum ROM_DMA_DONE_DETECT_METHOD	loadType	Defines whether the load function DMA should wait for the load to complete or return to the kernel

eOperation

Type of operation to perform

pRegs

Pointer to the base address of the DMA channel MMR registers

dataWidth

The maximum supported data width of the DMA channel. Used to configure the DMA_CFG.PSIZE PSIZE field in DMA_CFG

dstModifyMult

The modify multiplier, usually set to 1

pSource

Source Pointer used for transmit operations

pDestination

Destination Pointer used for receive operations

byteCount

Number of bytes to transfer

eDoneDetect

DMA Done Detection method used for the transfer

loadType

Defines whether the load function DMA should wait for the load to complete or return to the kernel

struct otp_data

Structure Type Declaration: `otp_data`

Container for accessing data to be written to OTP via the `adi_rom_otp_pgm()` routine.

Any pointers that are NULL will result in the object not being written. Any data values of 0 will not be written.

Table 40-66: `otp_data` Members

Type	Name	Description
<code>uint32_t (*)</code> [ROM_OTP_SZ_pll_calib]	<code>pll_calib</code>	Pointer to PLL calibration
<code>uint32_t</code>	<code>ospi_read_data_capture</code> : 16 (bit-field)	OSPI Read Data Capture register
<code>uint32_t (*)</code> [ROM_OTP_SZ_huk]	<code>huk</code>	Pointer to 256-bit Hardware Unique Key
<code>uint32_t (*)</code> [ROM_OTP_SZ_pvt_128key0]	<code>pvt_128key0</code>	Pointer to 128-bit AES Key
<code>uint32_t (*)</code> [ROM_OTP_SZ_pvt_128key1]	<code>pvt_128key1</code>	Pointer to 128-bit AES Key
<code>uint32_t (*)</code> [ROM_OTP_SZ_pvt_128key2]	<code>pvt_128key2</code>	Pointer to 128-bit AES Key
<code>uint32_t (*)</code> [ROM_OTP_SZ_pvt_128key3]	<code>pvt_128key3</code>	Pointer to 128-bit AES Key
<code>uint32_t (*)</code> [ROM_OTP_SZ_ek]	<code>ek</code>	Pointer to 256-bit endorsement key

Table 40-66: otp_data Members (Continued)

Type	Name	Description
uint32_t(*) [ROM_OTP_SZ_secure_emu_key]	secure_emu_key	Pointer to 128-bit Secure Debug Key
uint32_t	emu_key_disable: 16 (bitfield)	Secure emulation key loading into TAPC disable
uint32_t(*) [ROM_OTP_SZ_public_key0]	public_key0	Pointer to 512-bit public key used for boot stream authentication
uint32_t(*) [ROM_OTP_SZ_public_key1]	public_key1	Pointer to 512-bit public key used for boot stream authentication
uint32_t(*) [ROM_OTP_SZ_boot_info]	boot_info	Pointer to 512-bit boot customization structure, see also struct ADI_ROM_OTP_BOOT_INFO
uint8_t	antiroll_nv_cntr	Anti-rollback counter to prevent loads of older firmware during secure boot.
uint32_t(*) [ROM_OTP_SZ_gpl]	gpl	Pointer to 512-bit General purpose user space
uint32_t	bootModeDisable: 8 (bitfield)	Boot mode disable for permanently disabling specific boot modes
uint32_t(*) [ROM_OTP_SZ_preboot_ddr_cfg]	preboot_ddr_cfg	Pointer to 384-bit DMC Configuration. See also struct ADI_ROM_OTP_DMC_CONFIG
uint32_t(*) [ROM_OTP_SZ_stageID]	stageID	Pointer to 64-bit staging ID

pll_calib

Pointer to the PLL Calibration

ospi_read_data_capture

OSPI Read data capture in OTP space.

The mapping is as shown below:

```
typedef struct
{
    uint16_t readTiming:4; /*! Content for improving input setup time */
    uint16_t writeTiming:4; /*! Content for improving output hold time */
    uint16_t sampleEdgeSelection:1; /*! Sample edge selection for data output from flash */
    uint16_t reserved0:7;
}ADI_ROM_OSPI_READ_DATA_CAPTURE_CONFIG;
```

huk

Pointer to 256-bit Hardware Unique Key

pvt_128key0

Pointer to 128-bit AES Key

pvt_128key1

Pointer to 128-bit AES Key

pvt_128key2

Pointer to 128-bit AES Key

pvt_128key3

Pointer to 128-bit AES Key

ek

Pointer to 256-bit endorsement key

secure_emu_key

Pointer to 128-bit Secure Debug Key

emu_key_disable

Any non-zero value at this field will bypass the loading the Debug key into the TAPC register

public_key0

Pointer to 512-bit public key used for boot stream authentication

public_key1

Pointer to 512-bit public key used for boot stream authentication

boot_info

Pointer to 512-bit boot customization structure, see also [struct ADI_ROM_OTP_BOOT_INFO](#)

antiroll_nv_cntr

Anti-rollback counter to prevent loading of older firmware during secure boot.

The counter supports values of 0 through 31. The counter feature is disabled as long as the counter is set initially to 0.

gp1

Pointer to 512-bit General purpose user space

bootModeDisable

Boot mode disable for permanently disabling specific boot modes

preboot_dds_cfg

Pointer to 384-bit DMC Configuration. See also [struct ADI_ROM_OTP_DMC_CONFIG](#)

stageID

Pointer to 64-bit staging ID

Enumerations

The programming model for booting the processor uses the enumerations defined in this section.

enum ADI_ROM_BOOT_KEY_TYPE

Enumeration Type Declaration: `ADI_ROM_BOOT_KEY_TYPE`

Indicates if custom security keys are used for evaluation of secure boot.

By default the boot process fetches all security keys from OTP for use during secure boot. Custom security lets the program set their custom keys in the [struct ADI_ROM_BOOT_CONFIG](#) item (not from OTP). Secure boot is evaluated using the [adi_rom_Boot\(\)](#) function without provisioning keys in OTP.

Table 40-67: ADI_ROM_BOOT_KEY_TYPE Members

Enumerator	Description
ADI_ROM_CUSTOM_SECURITY	Enable use of custom security keys for authentication and decryption

ADI_ROM_CUSTOM_SECURITY

Enable use of custom security keys for authentication and decryption

enum ADI_ROM_BOOT_TYPE

Enumeration Type Declaration: `ADI_ROM_BOOT_TYPE`

Indicate to the boot kernel (in an open processor) if secure or non secure boot is required.

The boot kernel defaults to a secure boot unless the boot structure has been configured to indicate Non-Secure Boot

Table 40-68: ADI_ROM_BOOT_TYPE Members

Enumerator	Description
ADI_ROM_SECURE_BOOT_DIS	Non-Secure Boot
ADI_ROM_SECURE_BOOT	Secure Boot

ADI_ROM_SECURE_BOOT_DIS

Non-Secure Boot

ADI_ROM_SECURE_BOOT

Secure Boot

enum OTPCMD

Enumeration Type Declaration: OTPCMD

Commands required by the `adi_rom_otp_get()` routine to retrieve specific fields from the OTP memory.

Table 40-69: OTPCMD Members

Enumerator	Description
<code>otpcmd_reserved0</code>	Reserved
<code>otpcmd_pll_calib</code>	PLL Calibration
<code>otpcmd_ospi_read_data_capture</code>	OSPI Read data capture read/write timing field
<code>otpcmd_huk</code>	Hardware Unique Key
<code>otpcmd_pvt_128key0</code>	Customer Private AES Key0
<code>otpcmd_pvt_128key1</code>	Customer Private AES Key1
<code>otpcmd_pvt_128key2</code>	Customer Private AES Key2
<code>otpcmd_pvt_128key3</code>	Customer Private AES Key3
<code>otpcmd_ek</code>	Endorsement Key
<code>otpcmd_secure_emu_key</code>	Secure Emulation Key
<code>otpcmd_emu_key_disable</code>	Secure emulation key disable
<code>otpcmd_public_key0</code>	Customer Public Key0
<code>otpcmd_public_key1</code>	Customer Public Key1
<code>otpcmd_boot_info</code>	Customer Programmable Boot Information
<code>otpcmd_otpTiming</code>	OTP Read timing override
<code>otpcmd_antiroll_nv_cntr</code>	AntiRollback NV Counter
<code>otpcmd_gp1</code>	General Purpose 1
<code>otpcmd_bootModeDisable</code>	Boot Mode Disable Bits
<code>otpcmd_preboot_ddr_cfg</code>	User DMC configuration
<code>otpcmd_stageID</code>	StageID
<code>otpcmd_reserved1</code>	Reserved

otpcmd_reserved0

Reserved

otpcmd_pll_calib

PLL Calibration

otpcmd_ospi_read_data_capture

OSPI read data capture register read/write timing fields

otpcmd_huk

Hardware Unique Key

otpcmd_pvt_128key0

Customer Private AES Key0

otpcmd_pvt_128key1

Customer Private AES Key1

otpcmd_pvt_128key2

Customer Private AES Key2

otpcmd_pvt_128key3

Customer Private AES Key3

otpcmd_ek

Endorsement Key

otpcmd_secure_emu_key

Secure Emulation Key

otpcmd_emu_key_disable

Secure emulation key disable

otpcmd_public_key0

Customer Public Key0

otpcmd_public_key1

Customer Public Key1

otpcmd_boot_info

Customer Programmable Boot Information

otpcmd_otpTiming

OTP Read timing override

otpcmd_antiroll_nv_cntr

AntiRollback NV Counter

otpcmd_gp1

General Purpose 1

otpcmd_bootModeDisable

Boot Mode Disable Bits

otpcmd_preboot_ddr_cfg

User DMC configuration

otpcmd_stageID

StageID

enum ROM_BOOT_MDMA_CRC_SUPPORT

Enumeration Type Declaration: ROM_BOOT_MDMA_CRC_SUPPORT

MDMA Channel CRC support.

Specifies whether the MDMA channel supports CRC operations or not

Table 40-70: ROM_BOOT_MDMA_CRC_SUPPORT Members

Enumerator	Description
ROM_BOOT_DMA_CRC_SUPPORTED	MDMA Channel supports CRC
ROM_BOOT_DMA_CRC_NOT_SUPPORTED	MDMA Channel does not support CRC

ROM_BOOT_DMA_CRC_SUPPORTED

MDMA Channel supports CRC

ROM_BOOT_DMA_CRC_NOT_SUPPORTED

MDMA Channel does not support CRC

enum ROM_CORE_ID

Enumeration Type Declaration: ROM_CORE_ID

Core ID.

An enumeration for referencing a particular core

Table 40-71: ROM_CORE_ID Members

Enumerator	Description
ROM_CORE_ID0	Core 0
ROM_CORE_NUM_CORES	Number of Cores

ROM_CORE_ID0

Core 0

ROM_CORE_NUM_CORES

Number of Cores

enum ROM_DMA_DONE_DETECT_METHODEnumeration Type Declaration: `ROM_DMA_DONE_DETECT_METHOD`

DMA Done Detection Method.

Specifies the method used to detect the completion of the requested DMA operation.

When a program requests a non-blocking DMA operation, separate software is required to check the status of the DMA channel. The boot ROM does not provide an API for this operation.

NOTE: Trigger mode is not supported on this product

Table 40-72: ROM_DMA_DONE_DETECT_METHOD Members

Enumerator	Description
<code>ROM_DMA_DONE_NON_BLOCKING</code>	Return without waiting for the DMA to complete
<code>ROM_DMA_DONE_POLL_IRQDONE</code>	Poll on the IRQDONE bit in the DMA Status register
<code>ROM_DMA_DONE_WAKEUP_TRIGGER</code>	Configure a trigger to wakeup the core on completion

ROM_DMA_DONE_NON_BLOCKING

Return without waiting for the DMA to complete

ROM_DMA_DONE_POLL_IRQDONE

Poll on the IRQDONE bit in the DMA Status register

ROM_DMA_DONE_WAKEUP_TRIGGER

Configure a trigger to wakeup the core on completion

enum ROM_DMA_MDMA_IDEnumeration Type Declaration: `ROM_DMA_MDMA_ID`

MDMA Channel ID.

The ID of the Memory DMA channel. Used in the `struct ROM_DMA_MDMA_CONFIG` configuration to specify the Memory DMA channel to use for operations accessible using the `adi_rom_MemDma()` routine

Table 40-73: ROM_DMA_MDMA_ID Members

Enumerator	Description
ROM_DMA_MDMA0	Memory DMA Stream 0
ROM_DMA_MDMA1	Memory DMA Stream 1
ROM_DMA_MDMA2	Memory DMA Stream 2
ROM_DMA_MDMA3	Memory DMA Stream 3
ROM_DMA_MEMDMA_END_COUNT	Number of Memory DMA Streams

ROM_DMA_MDMA0

Memory DMA Stream 0

ROM_DMA_MDMA1

Memory DMA Stream 1

ROM_DMA_MDMA2

Memory DMA Stream 2

ROM_DMA_MDMA3

Memory DMA Stream 3

ROM_DMA_MEMDMA_END_COUNT

Number of Memory DMA Streams

enum ROM_DMA_MDMA_OPERATION

Enumeration Type Declaration: ROM_DMA_MDMA_OPERATION

MDMA Operation.

The operation determines if only an MDMA is required, or whether a CRC operation must be used in conjunction with the MDMA.

Table 40-74: ROM_DMA_MDMA_OPERATION Members

Enumerator	Description
ROM_DMA_MEM_COPY	Standard MDMA transfer from a source to a destination
ROM_DMA_MEM_CRC	Performs a CRC32 MDMA read operation and compares the result with an expected result
ROM_DMA_MEM_FILL	Uses the CRC peripheral to perform a fill operation with a 32-bit value
ROM_DMA_MEM_COMPARE	Uses the CRC peripheral to compare data with a constant 32-bit value
ROM_DMA_CRC_LUT_INIT	Initializes the CRC LUT from the supplied CRC Polynomial

ROM_DMA_MEM_COPY

Standard MDMA transfer from a source to a destination

ROM_DMA_MEM_CRC

Performs a CRC32 MDMA read operation and compares the result with an expected result

ROM_DMA_MEM_FILL

Uses the CRC peripheral to perform a fill operation with a 32-bit value

ROM_DMA_MEM_COMPARE

Uses the CRC peripheral to compare data with a constant 32-bit value

ROM_DMA_CRC_LUT_INIT

Initializes the CRC LUT from the supplied CRC Polynomial

enum ROM_DMA_PDMA_OPERATION

Enumeration Type Declaration: ROM_DMA_PDMA_OPERATION

Table 40-75: ROM_DMA_PDMA_OPERATION Members

Enumerator	Description
ROM_DMA_PERI_TX	Peripheral Transmit Operation
ROM_DMA_PERI_RX	Peripheral Receive Operation

ROM_DMA_PERI_TX

Peripheral Transmit Operation

ROM_DMA_PERI_RX

Peripheral Receive Operation

enum ROM_GETADDR_VALUE

Enumeration Type Declaration: ROM_GETADDR_VALUE

Parameter for `adi_rom_GetAddress()` function to retrieve the address of a data object stored in the boot ROM.

Table 40-76: ROM_GETADDR_VALUE Members

Enumerator	Description
ROM_GETADDR_CONSTANTS	Retrieve the address of the ROM_CONSTANTS_TYPE object
ROM_GETADDR_BMODE	Retrieve the address of the lookup table sorting the default <code>adi_rom_boot()</code> parameters for each boot mode

Table 40-76: ROM_GETADDR_VALUE Members (Continued)

Enumerator	Description
ROM_GETADDR_MDMAREGS	Retrieve the address of the <code>struct ROM_BOOT_MDMA_REGS</code> object
ROM_GETADDR_SPILOT	Retrieve the address of the lookup table in the ROM describing the various SPI master boot BCODE configurations
ROM_GETADDR_ECDSA_DOMAIN	Retrieve the address of the domain parameteres used for ECDSA

ROM_GETADDR_CONSTANTS

Retrieve the address of the `ROM_CONSTANTS_TYPE` object

ROM_GETADDR_BMODE

Retrieve the address of the lookup table sotring the default `adi_rom_boot()` parameters for each boot mode

ROM_GETADDR_MDMAREGS

Retrieve the address of the `struct ROM_BOOT_MDMA_REGS` object

ROM_GETADDR_SPILOT

Retrieve the address of the lookup table in the rom describing the various SPI master boot BCODE configurations

ROM_GETADDR_ECDSA_DOMAIN

Retrieve the address of the domain parameteres used for ECDSA

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

An optional hook routine is provided as a callback when calling a boot mode via `adi_rom_Boot`. This hook routine is called by the boot software first after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 40-77: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
ROM_HOOK_CALL_INIT_COMPLETE	Call was as a result of completion of the boot modes initialization function
ROM_HOOK_CALL_CONFIG_COMPLETE	Call was as a result of the completion of the boot modes configuration function
ROM_HOOK_REG_COMPLETE	Call was as a result of the completion of the boot modes pre-register initialization

enum ROM_SB_IMAGE_TYPE

Enumeration Type Declaration: ROM_SB_IMAGE_TYPE

Secure Boot Image Types.

The secure boot header contains a type field for the secure boot image type, this enumeration provides a complete list of all image types.

NOTE: The secure boot process does not necessarily support all image types defined.

Table 40-78: ROM_SB_IMAGE_TYPE Members

Enumerator	Description
ROM_SB_IMAGE_UNKNOWN	Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place
ROM_SB_IMAGE_BLP	Plain text BLP secure boot image supports authentication only with no decryption
ROM_SB_IMAGE_BLW	Keywrapped BLW secure boot image supports authentication and decryption, boot stream decryption key wrapped in the secure header
ROM_SB_IMAGE_BLE	Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header
ROM_SB_IMAGE_BLX	BLX Secure boot image supports authentication and decryption, boot stream decryption key wrapped located in OTP
ROM_SB_IMAGE_UNSUPPORTED	Used by software to indicate any other unsupported image type

ROM_SB_IMAGE_UNKNOWN

Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place

ROM_SB_IMAGE_BLP

Plain text BLP secure boot image supports authentication only with no decryption

ROM_SB_IMAGE_BLW

Keywrapped BLW secure boot image supports authentication and decryption, boot stream decryption key wrapped in the secure header

ROM_SB_IMAGE_BLE

Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header

ROM_SB_IMAGE_BLX

BLX Secure boot image supports authentication and decryption, boot stream decryption key wrapped located in OTP

ROM_SB_IMAGE_UNSUPPORTED

Used by software to indicate any other unsupported image type

41 System Crossbars (SCB)

A modern system on a chip (SoCs) contains multi-cores, memory controllers, security blocks, and other high speed peripherals. As system integration increases, SoCs need to provide bus connectivity that allows better throughput to reduce performance bottlenecks. While traditional point-to-point connection buses have performed well in smaller systems, there is a need to use advanced switching based bus architectures for efficient handling of data transfer between multiplicity of data sources and sinks in the system. Additionally, mixing various traffic types in the same SoC (for example control, communication over peripherals and computing) while sharing the same bus resources, create different requirements from the Quality of Service (QoS) perspective.

The system crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB architecture addresses the challenges described above. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable quality of service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.

SCB Features

The SCBs provide the following features:

- Efficient, pipelined bus transfer protocol for sustained throughput
- Full-duplex bus operation for flexibility and reduced latency
- Concurrent bus transfer support to allow multiple bus masters to access bus slaves simultaneously
- Protection model (privileged or secure) support for selective bus interconnect protection
- Simple priority-based QoS based arbitration model
- Programmable quality of service

SCB Functional Description

The following sections provide a functional description of the SCB.

- SCB Architectural Concepts

ADSP-2156x SCB0 Register List

Table 41-1: ADSP-2156x SCB0 Register List

Name	Description
SCB0_ACC_CH0_READ_QOS	Accelerator Channel 0 Read Quality of Service Register
SCB0_ACC_CH0_WRITE_QOS	Accelerator Channel 0 Write Quality of Service Register
SCB0_ACC_CH1_READ_QOS	Accelerator Channel 1 Read Quality of Service Register
SCB0_ACC_CH1_WRITE_QOS	Accelerator Channel 1 Write Quality of Service Register
SCB0_CRC0_CH0_READ_QOS	CRC0 Channel 0 Read Quality of Service Register
SCB0_CRC0_CH0_WRITE_QOS	CRC0 Channel 0 Write Quality of Service Register
SCB0_CRC0_CH1_READ_QOS	CRC0 Channel 1 Read Quality of Service Register
SCB0_CRC0_CH1_WRITE_QOS	CRC0 Channel 1 Write Quality of Service Register
SCB0_CRC1_CH0_READ_QOS	CRC1 Channel 0 Read Quality of Service Register
SCB0_CRC1_CH0_WRITE_QOS	CRC1 Channel 0 Write Quality of Service Register
SCB0_CRC1_CH1_READ_QOS	CRC1 Channel 1 Read Quality of Service Register
SCB0_CRC1_CH1_WRITE_QOS	CRC1 Channel 1 Write Quality of Service Register
SCB0_CRYPT0_READ_QOS	Crypto Read Quality of Service Register
SCB0_CRYPT0_WRITE_QOS	CRYPTO Write Quality of Service Register
SCB0_DBG_READ_QOS	DBG Read Quality of Service Register
SCB0_DBG_WRITE_QOS	DBG Write Quality of Service Register
SCB0_DLDMA0_CH0_READ_QOS	DLDMA0 Channel 0 Read Quality of Service Register
SCB0_DLDMA0_CH0_WRITE_QOS	DLDMA0 Channel 0 Write Quality of Service Register
SCB0_DLDMA0_CH1_READ_QOS	DLDMA0 Channel 1 Read Quality of Service Register
SCB0_DLDMA0_CH1_WRITE_QOS	DLDMA0 Channel 1 Write Quality of Service Register
SCB0_DLDMA1_CH0_READ_QOS	DLDMA1 Channel 0 Read Quality of Service Register
SCB0_DLDMA1_CH0_WRITE_QOS	DLDMA1 Channel 0 Write Quality of Service Register
SCB0_DLDMA1_CH1_READ_QOS	DLDMA1 Channel 1 Read Quality of Service Register
SCB0_DLDMA1_CH1_WRITE_QOS	DLDMA1 Channel 1 Write Quality of Service Register
SCB0_ETR_READ_QOS	ETR Read Quality of Service Register
SCB0_ETR_WRITE_QOS	ETR Write Quality of Service Register
SCB0_HSMDMA_CH0_READ_QOS	HSMDMA Channel 0 Read Quality of Service Register
SCB0_HSMDMA_CH0_WRITE_QOS	HSMDMA Channel 0 Write Quality of Service Register

Table 41-1: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_HSMDMA_CH1_READ_QOS	HSMDMA Channel 1 Read Quality of Service Register
SCB0_HSMDMA_CH1_WRITE_QOS	HSMDMA Channel 1 Write Quality of Service Register
SCB0_LP0_READ_QOS	LP0 Read Quality of Service Register
SCB0_LP0_WRITE_QOS	LP0 Write Quality of Service Register
SCB0_LP1_READ_QOS	LP1 Read Quality of Service Register
SCB0_LP1_WRITE_QOS	LP1 Write Quality of Service Register
SCB0_MLB_READ_QOS	MLB Read Quality of Service Register
SCB0_MLB_WRITE_QOS	MLB Write Quality of Service Register
SCB0_MSMDMA_CH0_READ_QOS	MSMDMA Channel 0 Read Quality of Service Register
SCB0_MSMDMA_CH0_WRITE_QOS	MSMDMA Channel 0 Write Quality of Service Register
SCB0_MSMDMA_CH1_READ_QOS	MSMDMA Channel 1 Read Quality of Service Register
SCB0_MSMDMA_CH1_WRITE_QOS	MSMDMA Channel 1 Write Quality of Service Register
SCB0_SH0_DPORT_READ_QOS	SH0 DPORT Read Quality of Service Register
SCB0_SH0_DPORT_WRITE_QOS	SH0 DPORT Write Quality of Service Register
SCB0_SH0_IPORT_READ_QOS	SH0 IPORT Read Quality of Service Register
SCB0_SH0_IPORT_WRITE_QOS	SH0 IPORT Write Quality of Service Register
SCB0_SH0_MMR_IB_READ_QOS	SH0 MMR Read Quality of Service Register
SCB0_SH0_MMR_IB_WRITE_QOS	SH0 MMR Write Quality of Service Register
SCB0_SP0A_READ_QOS	SP0A Read Quality of Service Register
SCB0_SP0A_WRITE_QOS	SP0A Write Quality of Service Register
SCB0_SP0B_READ_QOS	SP0B Read Quality of Service Register
SCB0_SP0B_WRITE_QOS	SP0B Write Quality of Service Register
SCB0_SP1A_READ_QOS	SP1A Read Quality of Service Register
SCB0_SP1A_WRITE_QOS	SP1A Write Quality of Service Register
SCB0_SP1B_READ_QOS	SP1B Read Quality of Service Register
SCB0_SP1B_WRITE_QOS	SP1B Write Quality of Service Register
SCB0_SP2A_READ_QOS	SP2A Read Quality of Service Register
SCB0_SP2A_WRITE_QOS	SP2A Write Quality of Service Register
SCB0_SP2B_READ_QOS	SP2B Read Quality of Service Register
SCB0_SP2B_WRITE_QOS	SP2B Write Quality of Service Register
SCB0_SP3A_READ_QOS	SP3A Read Quality of Service Register

Table 41-1: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_SP3A_WRITE_QOS	SP3A Write Quality of Service Register
SCB0_SP3B_READ_QOS	SP3B Read Quality of Service Register
SCB0_SP3B_WRITE_QOS	SP3B Write Quality of Service Register
SCB0_SP4A_READ_QOS	SP4A Read Quality of Service Register
SCB0_SP4A_WRITE_QOS	SP4A Write Quality of Service Register
SCB0_SP4B_READ_QOS	SP4B Read Quality of Service Register
SCB0_SP4B_WRITE_QOS	SP4B Write Quality of Service Register
SCB0_SP5A_READ_QOS	SP5A Read Quality of Service Register
SCB0_SP5A_WRITE_QOS	SP5A Write Quality of Service Register
SCB0_SP5B_READ_QOS	SP5B Read Quality of Service Register
SCB0_SP5B_WRITE_QOS	SP5B Write Quality of Service Register
SCB0_SP6A_READ_QOS	SP6A Read Quality of Service Register
SCB0_SP6A_WRITE_QOS	SP6A Write Quality of Service Register
SCB0_SP6B_READ_QOS	SP6B Read Quality of Service Register
SCB0_SP6B_WRITE_QOS	SP6B Write Quality of Service Register
SCB0_SP7A_READ_QOS	SP7A Read Quality of Service Register
SCB0_SP7A_WRITE_QOS	SP7A Write Quality of Service Register
SCB0_SP7B_READ_QOS	SP7B Read Quality of Service Register
SCB0_SP7B_WRITE_QOS	SP7B Write Quality of Service Register
SCB0_SPI0RX_READ_QOS	SPI0 RX Read Quality of Service Register
SCB0_SPI0RX_WRITE_QOS	SPI0 RX Write Quality of Service Register
SCB0_SPI0TX_READ_QOS	SPI0 TX Read Quality of Service Register
SCB0_SPI0TX_WRITE_QOS	SPI0 TX Write Quality of Service Register
SCB0_SPI1RX_READ_QOS	SPI1 RX Read Quality of Service Register
SCB0_SPI1RX_WRITE_QOS	SPI1 RX Write Quality of Service Register
SCB0_SPI1TX_READ_QOS	SPI1 TX Read Quality of Service Register
SCB0_SPI1TX_WRITE_QOS	SPI1 TX Write Quality of Service Register
SCB0_SPI2RX_READ_QOS	SPI2 RX Read Quality of Service Register
SCB0_SPI2RX_WRITE_QOS	SPI2 RX Write Quality of Service Register
SCB0_SPI2TX_READ_QOS	SPI2 TX Read Quality of Service Register
SCB0_SPI2TX_WRITE_QOS	SPI2 TX Write Quality of Service Register

Table 41-1: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_UART0_RX_READ_QOS	UART0 RX Read Quality of Service Register
SCB0_UART0_RX_WRITE_QOS	UART0 RX Write Quality of Service Register
SCB0_UART0_TX_READ_QOS	UART0 TX Read Quality of Service Register
SCB0_UART0_TX_WRITE_QOS	UART0 TX Write Quality of Service Register
SCB0_UART1_RX_READ_QOS	UART1 TX Read Quality of Service Register
SCB0_UART1_RX_WRITE_QOS	UART1 RX Write Quality of Service Register
SCB0_UART1_TX_READ_QOS	UART1 TX Read Quality of Service Register
SCB0_UART1_TX_WRITE_QOS	UART1 TX Write Quality of Service Register
SCB0_UART2_RX_READ_QOS	UART2 RX Read Quality of Service Register
SCB0_UART2_RX_WRITE_QOS	UART2 RX Write Quality of Service Register
SCB0_UART2_TX_READ_QOS	UART2 TX Read Quality of Service Register
SCB0_UART2_TX_WRITE_QOS	UART2 TX Write Quality of Service Register

ADSP-2156x SCB1 Register List

System Crossbar for DMC Memory Space

Table 41-2: ADSP-2156x SCB1 Register List

Name	Description
SCB1_MST_IB_SYNC_MODE	Master Synchronization Mode Register

ADSP-2156x SCB3 Register List

Table 41-3: ADSP-2156x SCB3 Register List

Name	Description
SCB3_APB_DCLK_0_IB_SYNC_MODE	APB DCLK 0 Synchronization Mode Register

SCB Architectural Concepts

This section describes the components of the SCB and the modules connected to it. The basic elements in the SCB are SCB masters, slaves master interfaces, and slave interfaces.

Masters

The controllers in the system that raise the data request in the form of a read/write transaction on the SCB are called masters. The system bus masters include peripheral Direct Memory Access (DMA) channels. These include the Serial Port (SPORT) DMA, and SPI DMAs, among others. Also included are the Memory-to-Memory DMA channels (MDMA), the L1 code fill block, and the processor cores.

Slaves

Slaves are SCB connections that are responding to transfer requests. Slaves include MMR registers, memory units, and various peripherals depending upon individual configurations. Each system slave has its own latencies and response times.

SCB Block Diagram

The SCB architectural model is shown in the *SCB Overview* figure. This figure shows a high level representation of a basic SCB connecting n Slaves to x Masters. A variable number of masters connect to a variable number of slaves in each SCB. In this example, all SIs connect to all MIs as indicated by the lines connecting them.

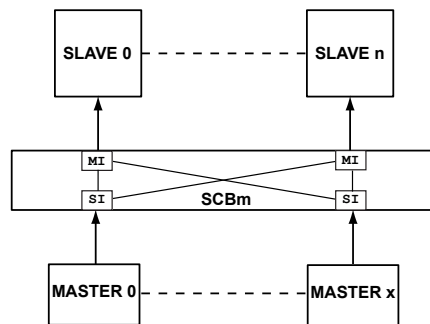


Figure 41-1: SCB Overview

Hierarchy Block Diagram

A system interconnect built from multiple SCBs in a hierarchical model is illustrated in the *SCB Hierarchy Overview* figure. The system master node level SCBs master connects multiple SIs to a single MI, which in turn connects to an SI of the system slave level node SCB.

As discussed above, all the masters in the system are distributed across different SCBs. A given SCB at system master node level connects directly to the system masters. These SCBs connect to SCB0 through its SIs forming a hierarchical structure. While a master has to access any slave, its first access goes through the SCB it is connected to, and then through SCB0, to reach its intended slave. This simplifies the connecting hardware in the basic SCB block by limiting the masters. Care must be taken when sharing masters to allow adequate throughput for their individual data transfer requirements.

In this example, all SIs are connected to all MIs.

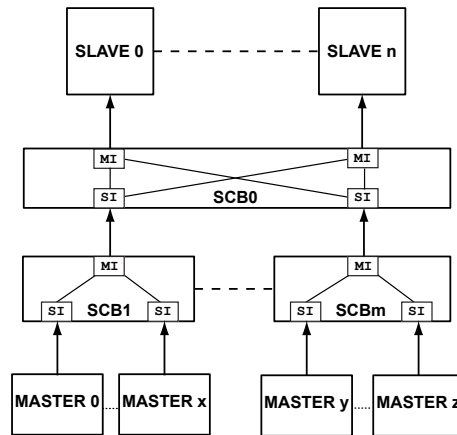


Figure 41-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the [SCB Block Diagram](#).

SCB Block Diagram

The figure shows the SCB block diagram.

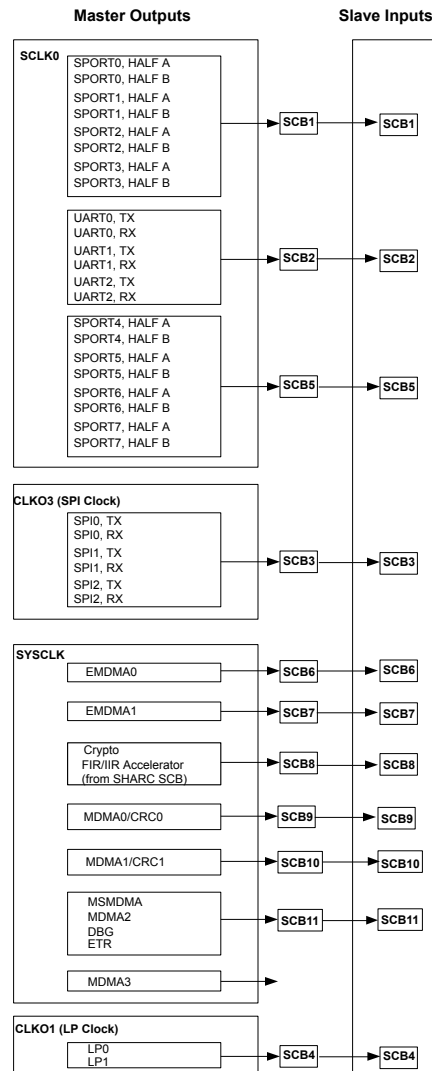


Figure 41-3: SCB Block Diagram

SHARC SCB with FIR/IIR Accelerator

The FIR/IIR accelerators on the ADSP-2156x processors are integrated with the SHARC+ core and operate at core clock (CCLK). This allows the FIR/IIR accelerator master ports to directly access the SHARC L1 memory with reduced latency as these accesses do not go through the main system fabric. SHARC+ core can also access the FIR/IIR accelerator MMR registers directly. A mini-SCB is added to support the SHARC SCB feature.

The figure shows the SHARC SCB block diagram.

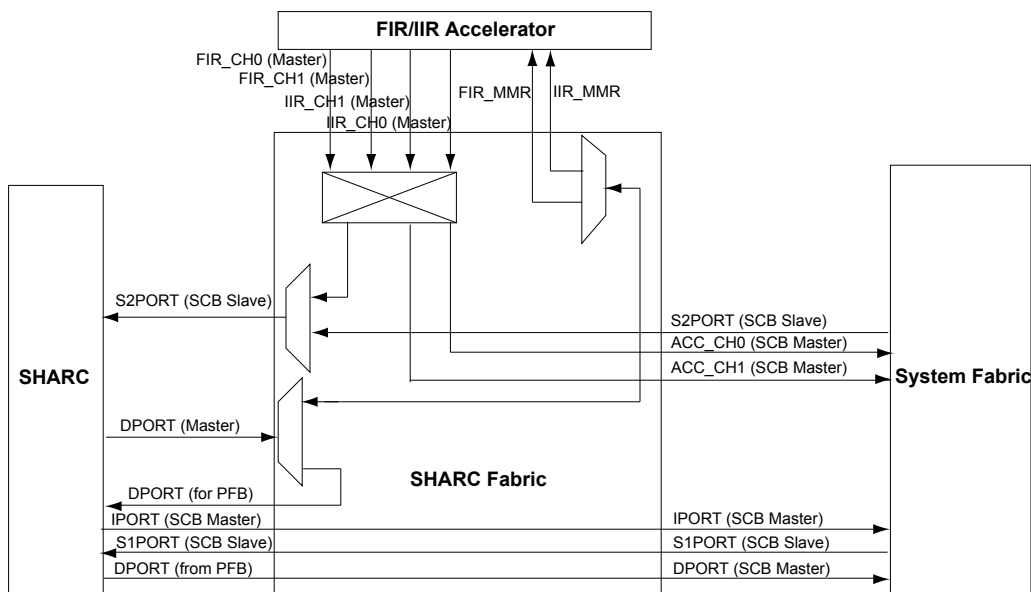


Figure 41-4: SHARC SCB with FIR/IIR Accelerator

Only SHARC+ core can access FIR/IIR accelerator MMRs. No other masters can access accelerator MMR space, including the DBG master. Debug can be done through SHARC+ core only. If the core hang, the accelerators cannot be debugged.

- SHARC+ core and FIR/IIR accelerators share the same security level.
- SHARC+ core’s MMR port is separated from data port. It is a separate master port for the main system fabric.
- Prefetch Buffer: After MMR split, the data port comes back into the SHARC+ core for PFB connection and goes to the main SCB via SHARC SCB with the clock domain crossing.

There are two types of peripherals that use DMA. The first have dedicated DMA channels controlled by the Dedicated DMA Engine (DDE) and have the same operating modes (see [DMA Operating Modes](#)) and use the same programming model ([DMA Channel Programming Model](#)). The second type is not controlled by the DDE. These peripherals have their own operating modes and programming models (see the peripheral chapter for this information). The peripheral types are shown in the *SCB-Controlled DMA Channel Peripherals* table.

Table 41-4: SCB Controlled DMA Channel Peripherals

	Masters	DDE DMA Channels	Non DDE DMA Channels
SCB1	SPORT0, HALF A	DMA0	
	SPORT0, HALF B	DMA1	
	SPORT1, HALF A	DMA2	
	SPORT1, HALF B	DMA3	
	SPORT2, HALF A	DMA4	
	SPORT2, HALF B	DMA5	

Table 41-4: SCB Controlled DMA Channel Peripherals (Continued)

	Masters	DDE DMA Channels	Non DDE DMA Channels
	SPORT3, HALF A	DMA6	
	SPORT3, HALF B	DMA7	
SCB2	UART0, TX	DMA20	
	UART0, RX	DMA21	
	UART1, TX	DMA34	
	UART1, RX	DMA35	
	UART2, TX	DMA37	
	UART2, RX	DMA38	
SCB3	SPI0, TX	DMA22	
	SPI0, RX	DMA23	
	SPI1, TX	DMA24	
	SPI1, RX	DMA25	
	SPI2, RX	DMA26	
	SPI2, RX	DMA27	
SCB4	LP0	DMA30	
	LP1	DMA36	
SCB5	SPORT4, HALF A	DMA10	
	SPORT4, HALF B	DMA11	
	SPORT5, HALF A	DMA12	
	SPORT5, HALF B	DMA13	
	SPORT6, HALF A	DMA14	
	SPORT6, HALF B	DMA15	
	SPORT7, HALF A	DMA16	
	SPORT7, HALF B	DMA17	
SCB6	EMDMA0	N/A	1 channel
SCB7	EMDMA1	N/A	1 channel
SCB8	CRYPTO	N/A	2 channels (1 read, 1 write)
	FIR/IIR Accelerator (from SHARC SCB)	N/A	1 channel
SCB9	CRC0/MDMA0	DMA8, DMA9	
SCB10	CRC1/MDMA1	DMA18, DMA19	

Table 41-4: SCB Controlled DMA Channel Peripherals (Continued)

	Masters	DDE DMA Channels	Non DDE DMA Channels
SCB11	MDMA2	DMA39, DMA40	
	DBG	N/A	
	ETR	N/A	
	MDMA3	DMA43, DMA44	

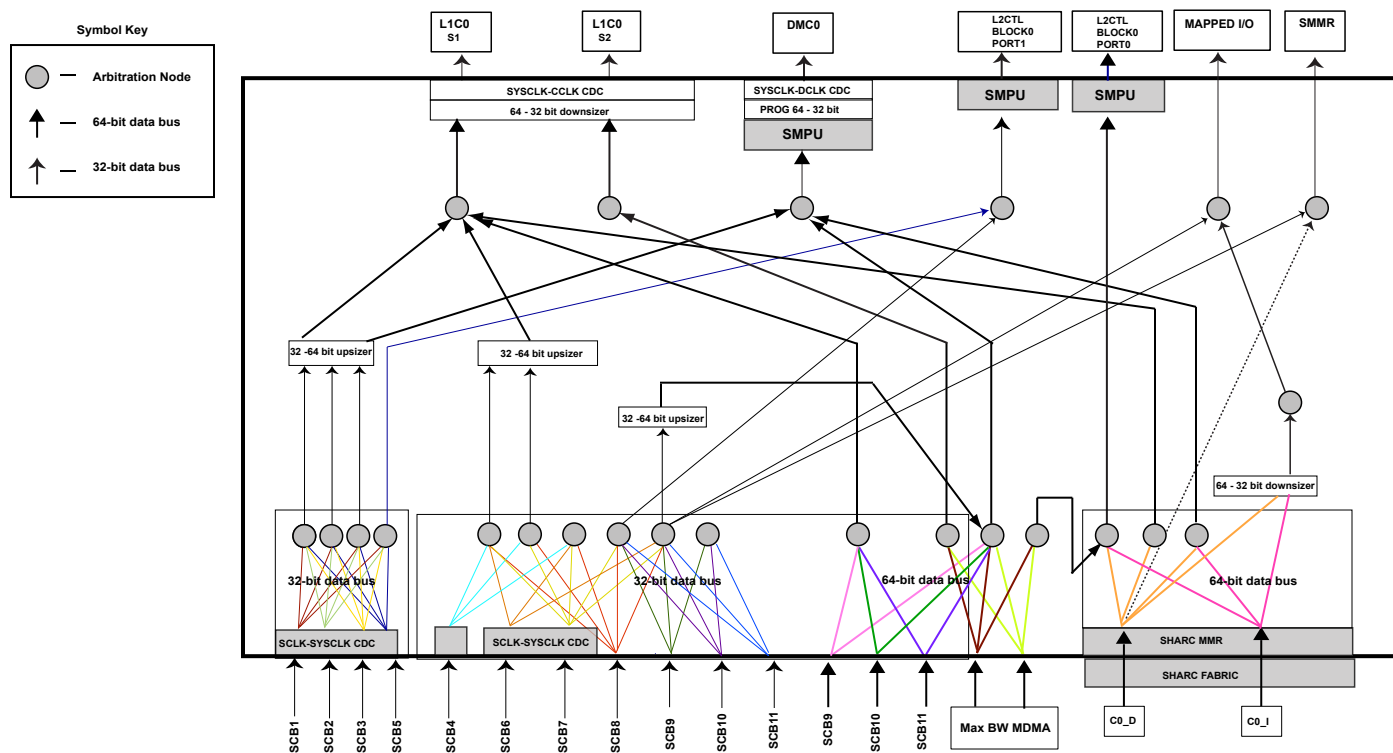


Figure 41-5: SCB Block Diagram - Interconnections

This figure is useful for the overview. It is also useful to observe the following relationships that are highlighted.

- The hierarchy of SCBs manages system bus interconnections, multiplexing, and arbitration among the peripherals on the processor.
- The SCBs connections support DMA channels for some peripherals and support dedicated connections for others. The connections also support memory-mapped register access for internal memory (L1 and L2) and for external memory.

The slave interface of the crossbar (where the masters such as DMA connect to) performs two functions. The first function is arbitration. SCBx handles arbitration. The second function is clock conversion. The programmable QoS registers can be viewed as being associated with the SCBx.

- Most of the peripherals and their SCBs are in the *SCLK0* domain. Crypto is also in the *SCLK0* domain. Enhanced bandwidth MDMA streams, Max BW MDMA, EMDMA0/1, Crypto, FIR/IIR accelerators, MLB, DBG, ETR, and their SCBs are in *SYSCLK* domain.
- Each peripheral has a latency for access across the SCB. The latency varies with the nature of the peripheral. Also, the number of active peripherals (especially for cases where multiple peripherals are active on a shared SCB) affects SCB performance. Refer to the [L2 System Memory](#) chapter for more details on the L2 memory organization.

MDMA channels are unidirectional. For example, MDMA SRC is read-only and MDMA is write-only. Access to the MMR space of SCB0 is allowed only in secure mode. The MMR space of SCB0 has registers for programming the QoS of various masters and controlling the clock domain crossing.

- Access to the QoS register space is allowed only in secure mode.
- CDC programmability is provided only for DDR clock domain. CDC for CCLK:SYSCLK is fixed to SYNC n:1, SCLK0:SYSCLK is fixed to 1:n, LPCLK:SYSCLK is fixed to m:n, and SPI clock:SYSCLK is fixed to m:n.
- S2 slave port of SHARCs is intended primarily for MDMA between the L1 of the 2 SHARCs. Hence, only the Max BW MDMA is given access to S2 ports while all others masters are given access to S1 port. This arrangement also means that S1 and S2 port now share the same address space as against the ADSP-2156x family of processors where an alias address space was maintained.
- SPIF is low bandwidth slave – sclk0/1 with 32 bit interface. Enhanced BW MDMA will be used to perform MDMA to them. Max BW MDMA is intended only for L1, L2 and DDR. (These slaves have the same bandwidth capability as Max BW MDMA).
- SPIF is a read-only flash. Peripherals are not able to read out of the SPI flash directly. Only MDMAs, crypto, cores, and debug tools are given access to it.

The following acronyms are used in the figure.

SCB0-11

Indicate SCB interfaces, connecting the system bus masters and slaves

SCLK0, SCLK1, SYSCLK

Indicate clock domains in which the specific SCBs operate. For more information on clock domains, see the Clock Generation Unit (CGU) chapter and the product data sheet.

CDC

Indicates the clock domain crossing

CO_D

Indicates the SHARC processor data bus (DM/PM)

C0_I

Indicates the SHARC processor instruction bus (PM)

L1C0 S1, L1C0 S2, L1C1 S1, L1C1 S2

Indicates the L1 memory for SHARC processor slave port (1 or 2)

DMC0

Indicates the dynamic memory controller (DMC) interfaces

SPO A/B, SP1 A/B

Indicates the serial port interfaces and their full-duplex halves

SPIO, SPI1 - RX/TX

Indicates the serial peripheral interfaces ports with receive or transmit paths

DBG, ETR

Indicates the Debug Port (DBG) and Embedded Trace Router (ETR) which provide access to debug and trace capabilities

SMPU

Indicates the system memory protection unit (SMPU)

MDMA0, MDMA1

Indicates the memory DMA 0 through 1 interfaces

SMMR

Indicates the system memory-mapped register interface

System Crossbars

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. The SCB (often referred to as the system interconnect fabric), is a collection of inter-connection units connecting system masters to slave memory spaces. The SCB connects one or more master devices to one or more memory-mapped slave devices. Each connected master can be a core that originates an SCB transaction, or a master interface of an upstream SCB cascaded interconnect. Each connected slave can be the final target of an SCB transaction or a slave interface of a downstream cascaded SCB interconnect (forming a hierarchy of SCBs).

Each SCB that has multiple masters and slaves share the total bandwidth of the SCB. (In a M:N configuration where M masters are connected to N slaves through the SCBx.)

The SCB provides separate channels for reads and writes. Read and write accesses through a given SCB do not share bandwidth. All the SCBs are 32 bits wide and run at SCLK speed, and can provide a bandwidth of up to 400 Mbytes per second for reads and writes separately (when SCLK = 100 MHz). Only SCB0, which is the major SCB in the SCB hierarchy, has the multiple paths between multiple master and slave interfaces.

See the [SCB Block Diagram](#).

All other SCBs in the chip connect to SCB0 through different slave interfaces. Other primary masters (DMAs, cores, and so on) in the system are distributed across these small SCBs. For a given SCB, all the master and slaves share the total bandwidth of the SCB. (Only SCB0 is the exception). Since different DMA channels are scattered across different SCBs (SCB1, SCB2 SCB3, and so on), they do not conflict for the bandwidth as long as they are in different SCB and are accessing different slaves. SCB0 allows for concurrent data transfer between multiple bus masters and multiple bus slaves, providing flexibility, and full-duplex operation.

If system accesses are carefully architected, SCB has a potential of providing sufficient sustained bandwidth in the end system.

Since the SCBs support burst transfers, it is important to configure the requesting master appropriately to make best use of available SCB bandwidth. For a DMA master, choosing the appropriate `DMA_CFG.MSIZE` value, is important from both a functional and a performance perspective. The value in the `DMA_CFG.PSIZE` bit field determines the width of the peripheral bus in use. It can be configured to 1-byte, 2-bytes, or 4-bytes. The `DMA_CFG.MSIZE` value determines the actual size of the SCB bus in use. It also determines the minimum number of bytes which are transferred from or to memory corresponding to a single DMA request or grant. The transfer can be 1-, 2-, 4-, 8-, 16-, or 32-bytes. If the `DMA_CFG.MSIZE` value is greater than the SCB bus width, the SCB performs burst transfers according to the width defined in `DMA_CFG.MSIZE`. When `DMA_CFG.MSIZE` is less than the SCB bus width, bursting is not supported and partial bus use results.

Each of the SCB unit in the fabric consists of N Slave interfaces (MSTn). Each of these interfaces has controls for read quality of service, write quality of service, and functional mode. A subset of these matrices includes controls for IB (Interface Block) sync mode, and bus functional mode. For more details on IB, see the clock domain synchronization section.

SCB Bus Master IDs

The SCB *Bus Master IDs* table indicates which masters are connected to each of the slave ports of SCB0. The tables also indicate the precise value of the ID as seen by the slave. These values are useful for SWU programming.

NOTE: For an overall diagram of all SCB interconnections, see the [SCB Block Diagram](#).

Table 41-5: Bus Master IDs

Master	Hex ID Values	Binary Values
DMA0 (SPORT0, HALF A)	0x0, 0x8	11'b0000000x000

Table 41-5: Bus Master IDs (Continued)

Master	Hex ID Values	Binary Values
DMA1 (SPORT0, HALF B)	0x1, 0x9	11'b0000000x001
DMA2 (SPORT1, HALF A)	0x2, 0xA	11'b0000000x010
DMA3 (SPORT1, HALF B)	0x3, 0xB	11'b0000000x011
DMA4 (SPORT2, HALF A)	0x4, 0xC	11'b0000000x100
DMA5 (SPORT2, HALF B)	0x5, 0xD	11'b0000000x101
DMA6 (SPORT3, HALF A)	0x6, 0xE	11'b0000000x110
DMA7 (SPORT3, HALF B)	0x7, 0xF	11'b0000000x111
DMA8 (Enh BW MDMA0 CRC, CH0)	0x86, 0x8E	11'b0001000x110
DMA9 (Enh BW MDMA0 CRC, CH1)	0x87, 0x8F	11'b0001000x111
MLB	0x104	11'b00100000100
DMA20 (UART0, TX)	0x180, 0x188	11'b0011000x000
DMA21 (UART0, RX)	0x184, 0x18C	11'b0011000x100
DMA37 (UART2, TX)	0x183, 0x18B	11'b0011000x011
DMA22 (SPI0, TX)	0x200, 0x208	11'b0100000x000
DMA23 (SPI0, RX)	0x201, 0x209	11'b0100000x001
DMA24 (SPI1, TX)	0x202, 0x20A	11'b0100000x010
DMA25 (SPI1, RX)	0x203, 0x20B	11'b0100000x011
DMA26 (SPI2, TX)	0x205, 0x20D	11'b0100000x101
DMA27 (SPI2, RX)	0x204, 0x20C	11'b0100000x100
DMA30 (LP0)	0x280, 0x288	11'b0101000x000
DMA34 (UART1, TX)	0x181, 0x189	11'b0011000x001
DMA35 (UART1, RX)	0x182, 0x18A	11'b0011000x010
DMA36 (LP1)	0x281, 0x289	11'b0101000x001
CRYPTO	0x302	11'b01100000010
ACC (CH0)	There are 32 values. "x" is either a 0 or 1.	11'b0110xxxx000
ACC (CH1)	There are 32 values. "x" is either a 0 or 1.	11'b0110xxxx001
EMDMA0 (CH0)	0x380	11'b01110000000
EMDMA0 (CH1)	0x381	11'b01110000001
EMDMA1 (CH0)	0x401	11'b10000000001
EMDMA1 (CH1)	0x400	11'b10000000000
DMA39 (Enh BW MDMA2, CH0)	0x480, 0x488	11'b1001000x000

Table 41-5: Bus Master IDs (Continued)

Master	Hex ID Values	Binary Values
DMA40 (Enh BW MDMA2, CH1)	0x481, 0x489	11'b1001000x001
DBG	0x482	11'b10010000010
ETR	0x483	11'b10010000011
DMA18 (Enh BW MDMA1 CRC1, CH0)	0x504, 0x50C	11'b1010000x100
DMA19 (Enh BW MDMA1 CRC1, CH1)	0x505, 0x50D	11'b1010000x101
DMA38 (UART2, RX)	0x185, 0x18D	11'b0011000x101
SH0 (DPORT)	There are 32 values. "x" is either a 0 or 1.	11'b1011xxxx000
SH0 (IPOINT)	0x581	11'b10110000001
DMA43 (Enh BW MDMA3, CH0)	0x600, 0x608	11'b1100000x000
DMA44 (Enh BW MDMA3, CH1)	0x601, 0x609	11'b1100000x001
DMA10 (SPORT4, HALFA)	0x680, 0x688	11'b1101000x000
DMA11 (SPORT4, HALFB)	0x681, 0x689	11'b1101000x001
DMA12 (SPORT5, HALFA)	0x682, 0x68A	11'b1101000x010
DMA13 (SPORT5, HALFB)	0x683, 0x68B	11'b1101000x011
DMA14 (SPORT6, HALFA)	0x684, 0x68C	11'b1101000x100
DMA15 (SPORT6, HALFB)	0x685, 0x68D	11'b1101000x101
DMA16 (SPORT7, HALFA)	0x686, 0x68E	11'b1101000x110
DMA17 (SPORT7, HALFB)	0x687, 0x68F	11'b1101000x111
SH0_MMR	There are 32 values. "x" is either a 0 or 1.	11'b1110xxxx000

SCB Programming Model

The following sections provide information for programming the SCB properly.

Programming SCB Arbitration

Each slave interface has a QoS value (priority) associated with both read and write channels. These values are 4 bits present in the `SCB0_MSTx_RQOS` and `SCB0_MSTx_WQOS` registers. At the entry point to the infrastructure, all transactions are allocated this programmable local QoS value. The arbitration of the transaction throughout the infrastructure uses this QoS. At any arbitration node, a fixed priority exists for transactions with a different QoS. The highest value has the highest priority.

If there are coincident transactions at an arbitration node with the same QoS that require arbitration, then the network uses a Least Recently Granted (LRG) algorithm. At each switch, the master with the highest QoS acquires access and that switch output takes the QoS value of the winner for that transaction. At the next switch slave interface, the master uses the QoS value of the winner. QoS can have values from 0 (lowest priority) to 15 (highest priority).

For example in the following figure, *SCB Arbitration*:

1. At SCB1, masters (1, 2, 3) have RQOS values of (6, 4, 2)
2. At SCB2, masters (4, 5, 6) have RQOS values of (12, 13, 1)

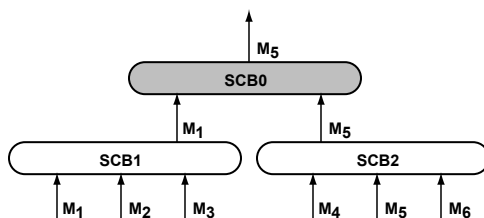


Figure 41-6: SCB Arbitration

In this case, master 1 wins at SCB1, and master 5 wins at SCB2. However, in a perfect competition at SCB0, masters 4 and 5 had the highest overall RQOS values. Masters 4 and 5 would have fought for arbitration directly at SCB0. However, because of the mini-SCBs, master 1, at a much lower RQOS value, is able to win against master 4 and make it all the way to SCB0.

Programming Clock Domain Crossing Registers

The clock domain crossings in the fabric are defined as follows:

- CCLK0:SYSCLK SYNC n:1 (SHARC core clock)
- SCLK0:SYSCLK SYNC 1:n
- LPCLK:SYSCLK m:n
- DCLK:SYSCLK Programmable
- SPICLK: SYSCLK m:n

The DCLK:SYSCLK clock domain crossing is the only one defined as programmable. Use the [SCB1_MST_IB_SYNC_MODE](#) and [SCB3_APB_DCLK_0_IB_SYNC_MODE](#) registers to program DCLK:SYSCLK in the DMC CDC fabric and MMRG fabric, respectively. The DMC CDC fabric handles clock domain conversion for data accesses and the MMRG fabric handles the clock domain conversion for MMR accesses. Both of these registers must be programmed based on the DCLK:SYSCLK ratio.

The *Sync Mode Bit Field Description* table describes the sync modes.

Table 41-6: Sync Mode Bit Field Description

Sync Mode	Description
0	sync 1:1
1	sync n:1
2	sync 1:n

Table 41-6: Sync Mode Bit Field Description (Continued)

Sync Mode	Description
3	sync m:n
4	async

To change the clock domain crossing mode, follow the actions described in the *Changing Clock Domain Crossing Modes* table.

Table 41-7: Changing Clock Domain Crossing Modes

Original Mode	Required Mode	Action
ASYNC	Any other mode	Change the clocks then change the MMR register
Any mode	ASYNC	Change the MMR register then change clocks to ASYNC.
m:n	1:1	Change the clocks, then change the register.
1:1	m:n	Change the register, then change the clocks.

SCB Programming Concepts

The SCB arbitration model among master or slave SCBs of the processor is fixed (not programmable). But, each slave does have a quality of service (QoS) programmable feature that affects arbitration.

The arbitration of transactions in SCB is based on the QoS value or the priority of the transaction. All masters with the same priority form a priority group. Arbitration is granted to the highest priority group from which a member is trying to win access, and within that group, to the highest master at that time. When a master wins arbitration, it is relegated to the bottom of its group to ensure that it cannot prevent other masters in its group from accessing the slave.

If you configure all master priorities to different levels, the arbiter implements a fixed priority scheme. This scheme occurs because each master is in a group of its own, and therefore, masters maintain their ordering.

The LRG and fixed priority modes concurrently exist when the master priority value registers are programmed with a combination of identical and unique values.

NOTE: The SCB arbitration hierarchy is fixed (for example, SCB1 master to SCB1 slave). However, multiple master inputs to the same slave permit QoS programming.

The *LRG Arbitration Example* figure shows three groups with different QoS values. Masters in the same group share a QoS value. The arbitration occurs using an LRG scheme.

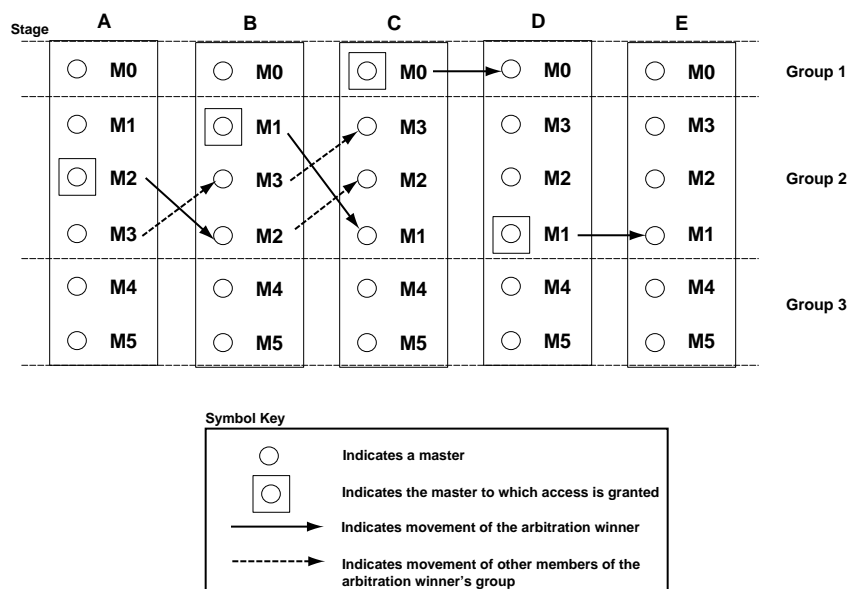


Figure 41-7: LRG Arbitration Example

The QoS value assigned to a transaction at entry point is carried forward by the transaction as it passes through all arbitration stages in the SCB. QoS for all masters is configured as programmable in the system fabric interconnect.

The priority of the masters fits into three groups:

- Group A - Peripherals with an external interface, without flow control and with real-time processing requirements
- Group B - Core
- Group C - Peripherals with flow control, MDMAs, and offload engines

Group A: Peripherals with external interface, without flow control and with real-time processing requirements

These masters are assigned the highest priority. They are latency-critical – a large increase in latency could potentially result in data corruption and catastrophic failure. This high priority SCB group includes masters connected to SCB1, SCB2, SCB3, and SCB5. Masters in this group are assigned the QoS reset value of 12/10.

Group B: Core

Core is latency-sensitive. Once a core requests data, it typically waits for the data without performing anything else in parallel. An increase in the latency has a direct impact on the performance of the core. Core is assigned medium priority and a QoS reset value of 7.

Group C: Peripherals with flow control, MDMAs, and offload engines

These masters are assigned the lowest priority. Peripherals with flow control can tolerate some increase in latency. MDMAs and offload engines are data intensive and do not have any external interface. Masters in this group are assigned the QoS reset value of 3/1. The Group C masters are connected to SCB4, SCB6, SCB7, SCB8, SCB9, SCB10, and SCB11.

Table 41-8: QoS Register Table

Master ID	Master	read_qos Reset Value	write_qos Reset Value
1	SPORT0_A_DMA	12	12
2	SPORT0_B_DMA	12	12
3	SPORT1_A_DMA	12	12
4	SPORT1_B_DMA	12	12
5	SPORT2_A_DMA	12	12
6	SPORT2_B_DMA	12	12
7	SPORT3_A_DMA	12	12
8	SPORT3_B_DMA	12	12
9	SPORT4_A_DMA	12	12
10	SPORT4_B_DMA	12	12
11	SPORT5_A_DMA	12	12
12	SPORT5_B_DMA	12	12
13	SPORT6_A_DMA	12	12
14	SPORT6_B_DMA	12	12
15	SPORT7_A_DMA	12	12
16	SPORT7_B_DMA	12	12
17	SPI0TX	12	12
18	SPI0RX	12	12
19	SPI1TX	12	12
20	SPI1RX	12	12
21	SPI2TX	12	12
22	SPI2RX	12	12
23	MLB	12	12
24	UART0_TX	10	10
25	UART0_RX	10	10
26	UART1_TX	10	10
27	UART1_RX	10	10

Table 41-8: QoS Register Table (Continued)

Master ID	Master	read_qos Reset Value	write_qos Reset Value
28	UART2_TX	10	10
29	UART2_RX	10	10
30	SH0_DPORT	7	7
31	SH0_IPORT	7	7
32	SH0_MMR	7	7
33	DBG	3	3
34	ETR	3	3
35	EMDMA0_CH0	1	1
36	EMDMA0_CH1	1	1
37	EMDMA1_CH0	1	1
38	EMDMA1_CH1	1	1
39	LP0	1	1
40	LP1	1	1
41	CRYPTO	1	1
42	ACC_CH0	1	1
43	ACC_CH1	1	1
44	MDMA0_CH0	1	1
45	MDMA0_CH1	1	1
46	MDMA1_CH0	1	1
47	MDMA1_CH1	1	1
48	MDMA2_CH0	1	1
49	MDMA2_CH1	1	1
50	MDMA3_CH0	1	1
51	MDMA3_CH1	1	1

QoS Programming

Adhere to the following guidelines if software is modifying the reset value of QoS:

- The highest QoS of any master in a low priority SCB group must be less than the lowest QoS of any master in a medium priority SCB group
- The highest QoS of any master in medium priority SCB group must be less than the lowest QoS of any master in high priority SCB group

Reset values of QoS have been spaced out to allow software to lower or higher the priority of a few masters without having to modify the priority all the masters. For example, the QoS of the LP can be reduced from 12 to 11 without violating the guidelines.

ADSP-2156x SCB0 Register Descriptions

(SCB0) contains the following registers.

Table 41-9: ADSP-2156x SCB0 Register List

Name	Description
SCB0_ACC_CH0_READ_QOS	Accelerator Channel 0 Read Quality of Service Register
SCB0_ACC_CH0_WRITE_QOS	Accelerator Channel 0 Write Quality of Service Register
SCB0_ACC_CH1_READ_QOS	Accelerator Channel 1 Read Quality of Service Register
SCB0_ACC_CH1_WRITE_QOS	Accelerator Channel 1 Write Quality of Service Register
SCB0_CRC0_CH0_READ_QOS	CRC0 Channel 0 Read Quality of Service Register
SCB0_CRC0_CH0_WRITE_QOS	CRC0 Channel 0 Write Quality of Service Register
SCB0_CRC0_CH1_READ_QOS	CRC0 Channel 1 Read Quality of Service Register
SCB0_CRC0_CH1_WRITE_QOS	CRC0 Channel 1 Write Quality of Service Register
SCB0_CRC1_CH0_READ_QOS	CRC1 Channel 0 Read Quality of Service Register
SCB0_CRC1_CH0_WRITE_QOS	CRC1 Channel 0 Write Quality of Service Register
SCB0_CRC1_CH1_READ_QOS	CRC1 Channel 1 Read Quality of Service Register
SCB0_CRC1_CH1_WRITE_QOS	CRC1 Channel 1 Write Quality of Service Register
SCB0_CRYPT0_READ_QOS	Crypto Read Quality of Service Register
SCB0_CRYPT0_WRITE_QOS	CRYPTO Write Quality of Service Register
SCB0_DBG_READ_QOS	DBG Read Quality of Service Register
SCB0_DBG_WRITE_QOS	DBG Write Quality of Service Register
SCB0_DLDMA0_CH0_READ_QOS	DLDMA0 Channel 0 Read Quality of Service Register
SCB0_DLDMA0_CH0_WRITE_QOS	DLDMA0 Channel 0 Write Quality of Service Register
SCB0_DLDMA0_CH1_READ_QOS	DLDMA0 Channel 1 Read Quality of Service Register
SCB0_DLDMA0_CH1_WRITE_QOS	DLDMA0 Channel 1 Write Quality of Service Register
SCB0_DLDMA1_CH0_READ_QOS	DLDMA1 Channel 0 Read Quality of Service Register
SCB0_DLDMA1_CH0_WRITE_QOS	DLDMA1 Channel 0 Write Quality of Service Register
SCB0_DLDMA1_CH1_READ_QOS	DLDMA1 Channel 1 Read Quality of Service Register
SCB0_DLDMA1_CH1_WRITE_QOS	DLDMA1 Channel 1 Write Quality of Service Register
SCB0_ETR_READ_QOS	ETR Read Quality of Service Register

Table 41-9: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_ETR_WRITE_QOS	ETR Write Quality of Service Register
SCB0_HSMDMA_CH0_READ_QOS	HSMDMA Channel 0 Read Quality of Service Register
SCB0_HSMDMA_CH0_WRITE_QOS	HSMDMA Channel 0 Write Quality of Service Register
SCB0_HSMDMA_CH1_READ_QOS	HSMDMA Channel 1 Read Quality of Service Register
SCB0_HSMDMA_CH1_WRITE_QOS	HSMDMA Channel 1 Write Quality of Service Register
SCB0_LP0_READ_QOS	LP0 Read Quality of Service Register
SCB0_LP0_WRITE_QOS	LP0 Write Quality of Service Register
SCB0_LP1_READ_QOS	LP1 Read Quality of Service Register
SCB0_LP1_WRITE_QOS	LP1 Write Quality of Service Register
SCB0_MLB_READ_QOS	MLB Read Quality of Service Register
SCB0_MLB_WRITE_QOS	MLB Write Quality of Service Register
SCB0_MSMDMA_CH0_READ_QOS	MSMDMA Channel 0 Read Quality of Service Register
SCB0_MSMDMA_CH0_WRITE_QOS	MSMDMA Channel 0 Write Quality of Service Register
SCB0_MSMDMA_CH1_READ_QOS	MSMDMA Channel 1 Read Quality of Service Register
SCB0_MSMDMA_CH1_WRITE_QOS	MSMDMA Channel 1 Write Quality of Service Register
SCB0_SH0_DPORT_READ_QOS	SH0 DPORT Read Quality of Service Register
SCB0_SH0_DPORT_WRITE_QOS	SH0 DPORT Write Quality of Service Register
SCB0_SH0_IPORT_READ_QOS	SH0 IPORT Read Quality of Service Register
SCB0_SH0_IPORT_WRITE_QOS	SH0 IPORT Write Quality of Service Register
SCB0_SH0_MMR_IB_READ_QOS	SH0 MMR Read Quality of Service Register
SCB0_SH0_MMR_IB_WRITE_QOS	SH0 MMR Write Quality of Service Register
SCB0_SP0A_READ_QOS	SP0A Read Quality of Service Register
SCB0_SP0A_WRITE_QOS	SP0A Write Quality of Service Register
SCB0_SP0B_READ_QOS	SP0B Read Quality of Service Register
SCB0_SP0B_WRITE_QOS	SP0B Write Quality of Service Register
SCB0_SP1A_READ_QOS	SP1A Read Quality of Service Register
SCB0_SP1A_WRITE_QOS	SP1A Write Quality of Service Register
SCB0_SP1B_READ_QOS	SP1B Read Quality of Service Register
SCB0_SP1B_WRITE_QOS	SP1B Write Quality of Service Register
SCB0_SP2A_READ_QOS	SP2A Read Quality of Service Register
SCB0_SP2A_WRITE_QOS	SP2A Write Quality of Service Register

Table 41-9: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_SP2B_READ_QOS	SP2B Read Quality of Service Register
SCB0_SP2B_WRITE_QOS	SP2B Write Quality of Service Register
SCB0_SP3A_READ_QOS	SP3A Read Quality of Service Register
SCB0_SP3A_WRITE_QOS	SP3A Write Quality of Service Register
SCB0_SP3B_READ_QOS	SP3B Read Quality of Service Register
SCB0_SP3B_WRITE_QOS	SP3B Write Quality of Service Register
SCB0_SP4A_READ_QOS	SP4A Read Quality of Service Register
SCB0_SP4A_WRITE_QOS	SP4A Write Quality of Service Register
SCB0_SP4B_READ_QOS	SP4B Read Quality of Service Register
SCB0_SP4B_WRITE_QOS	SP4B Write Quality of Service Register
SCB0_SP5A_READ_QOS	SP5A Read Quality of Service Register
SCB0_SP5A_WRITE_QOS	SP5A Write Quality of Service Register
SCB0_SP5B_READ_QOS	SP5B Read Quality of Service Register
SCB0_SP5B_WRITE_QOS	SP5B Write Quality of Service Register
SCB0_SP6A_READ_QOS	SP6A Read Quality of Service Register
SCB0_SP6A_WRITE_QOS	SP6A Write Quality of Service Register
SCB0_SP6B_READ_QOS	SP6B Read Quality of Service Register
SCB0_SP6B_WRITE_QOS	SP6B Write Quality of Service Register
SCB0_SP7A_READ_QOS	SP7A Read Quality of Service Register
SCB0_SP7A_WRITE_QOS	SP7A Write Quality of Service Register
SCB0_SP7B_READ_QOS	SP7B Read Quality of Service Register
SCB0_SP7B_WRITE_QOS	SP7B Write Quality of Service Register
SCB0_SPI0RX_READ_QOS	SPI0 RX Read Quality of Service Register
SCB0_SPI0RX_WRITE_QOS	SPI0 RX Write Quality of Service Register
SCB0_SPI0TX_READ_QOS	SPI0 TX Read Quality of Service Register
SCB0_SPI0TX_WRITE_QOS	SPI0 TX Write Quality of Service Register
SCB0_SPI1RX_READ_QOS	SPI1 RX Read Quality of Service Register
SCB0_SPI1RX_WRITE_QOS	SPI1 RX Write Quality of Service Register
SCB0_SPI1TX_READ_QOS	SPI1 TX Read Quality of Service Register
SCB0_SPI1TX_WRITE_QOS	SPI1 TX Write Quality of Service Register
SCB0_SPI2RX_READ_QOS	SPI2 RX Read Quality of Service Register

Table 41-9: ADSP-2156x SCB0 Register List (Continued)

Name	Description
SCB0_SPI2RX_WRITE_QOS	SPI2 RX Write Quality of Service Register
SCB0_SPI2TX_READ_QOS	SPI2 TX Read Quality of Service Register
SCB0_SPI2TX_WRITE_QOS	SPI2 TX Write Quality of Service Register
SCB0_UART0_RX_READ_QOS	UART0 RX Read Quality of Service Register
SCB0_UART0_RX_WRITE_QOS	UART0 RX Write Quality of Service Register
SCB0_UART0_TX_READ_QOS	UART0 TX Read Quality of Service Register
SCB0_UART0_TX_WRITE_QOS	UART0 TX Write Quality of Service Register
SCB0_UART1_RX_READ_QOS	UART1 TX Read Quality of Service Register
SCB0_UART1_RX_WRITE_QOS	UART1 RX Write Quality of Service Register
SCB0_UART1_TX_READ_QOS	UART1 TX Read Quality of Service Register
SCB0_UART1_TX_WRITE_QOS	UART1 TX Write Quality of Service Register
SCB0_UART2_RX_READ_QOS	UART2 RX Read Quality of Service Register
SCB0_UART2_RX_WRITE_QOS	UART2 RX Write Quality of Service Register
SCB0_UART2_TX_READ_QOS	UART2 TX Read Quality of Service Register
SCB0_UART2_TX_WRITE_QOS	UART2 TX Write Quality of Service Register

Accelerator Channel 0 Read Quality of Service Register

The `SCB0_ACC_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

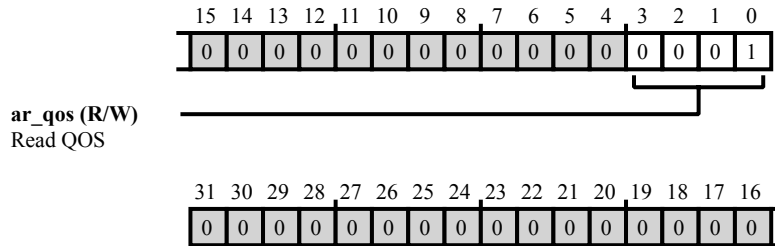


Figure 41-8: SCB0_ACC_CH0_READ_QOS Register Diagram

Table 41-10: SCB0_ACC_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

Accelerator Channel 0 Write Quality of Service Register

The `SCB0_ACC_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

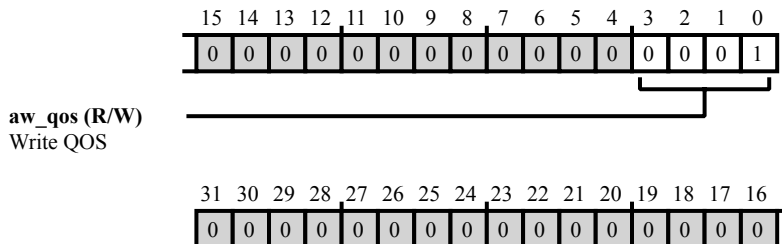


Figure 41-9: SCB0_ACC_CH0_WRITE_QOS Register Diagram

Table 41-11: SCB0_ACC_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

Accelerator Channel 1 Read Quality of Service Register

The `SCB0_ACC_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

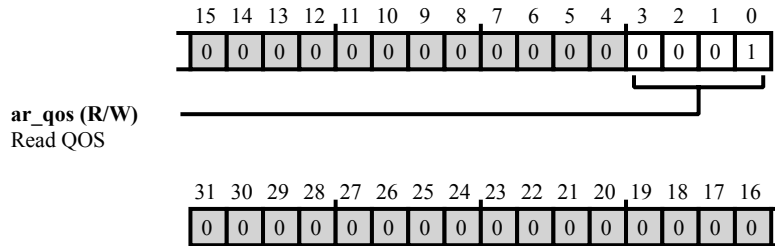


Figure 41-10: `SCB0_ACC_CH1_READ_QOS` Register Diagram

Table 41-12: `SCB0_ACC_CH1_READ_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

Accelerator Channel 1 Write Quality of Service Register

The `SCB0_ACC_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

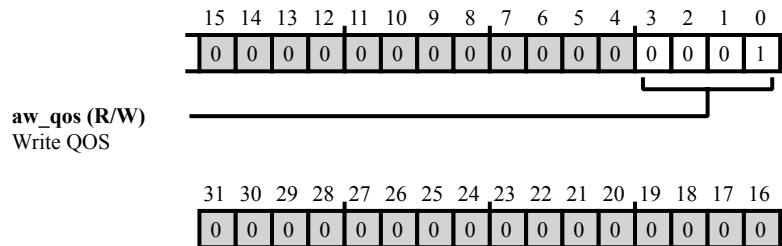


Figure 41-11: SCB0_ACC_CH1_WRITE_QOS Register Diagram

Table 41-13: SCB0_ACC_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

CRC0 Channel 0 Read Quality of Service Register

The `SCB0_CRC0_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

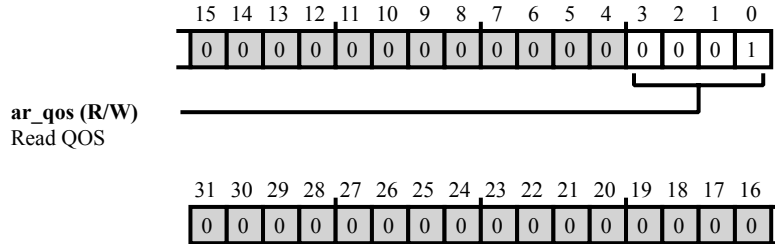


Figure 41-12: SCB0_CRC0_CH0_READ_QOS Register Diagram

Table 41-14: SCB0_CRC0_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

CRC0 Channel 0 Write Quality of Service Register

The `SCB0_CRC0_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

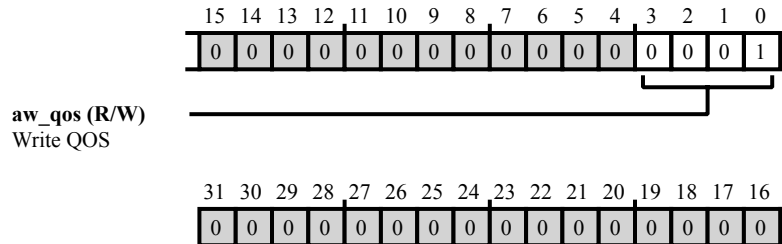


Figure 41-13: SCB0_CRC0_CH0_WRITE_QOS Register Diagram

Table 41-15: SCB0_CRC0_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

CRC0 Channel 1 Read Quality of Service Register

The `SCB0_CRC0_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

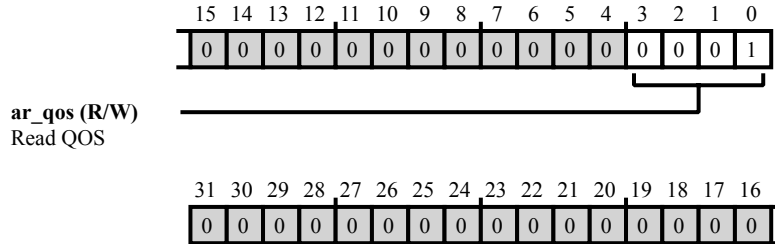


Figure 41-14: SCB0_CRC0_CH1_READ_QOS Register Diagram

Table 41-16: SCB0_CRC0_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

CRC0 Channel 1 Write Quality of Service Register

The `SCB0_CRC0_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

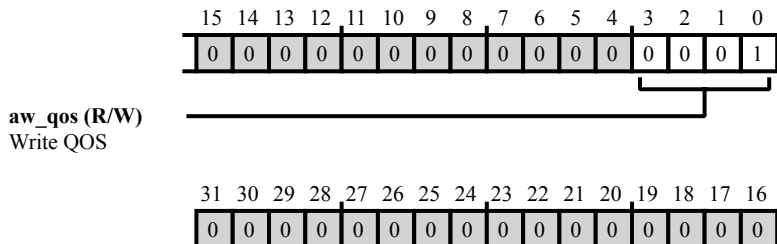


Figure 41-15: SCB0_CRC0_CH1_WRITE_QOS Register Diagram

Table 41-17: SCB0_CRC0_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

CRC1 Channel 0 Read Quality of Service Register

The `SCB0_CRC1_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

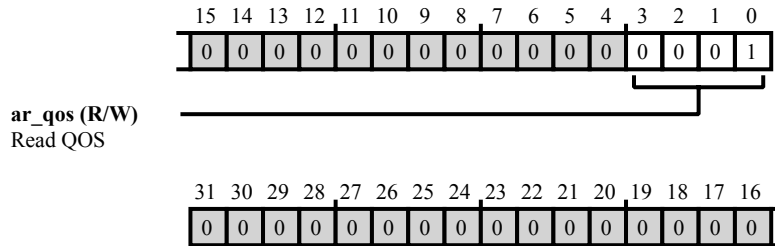


Figure 41-16: SCB0_CRC1_CH0_READ_QOS Register Diagram

Table 41-18: SCB0_CRC1_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

CRC1 Channel 0 Write Quality of Service Register

The `SCB0_CRC1_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

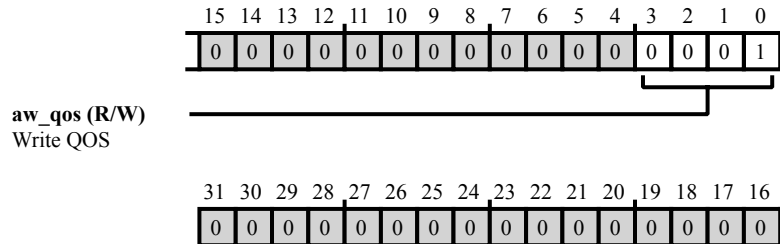


Figure 41-17: SCB0_CRC1_CH0_WRITE_QOS Register Diagram

Table 41-19: SCB0_CRC1_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

CRC1 Channel 1 Read Quality of Service Register

The `SCB0_CRC1_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

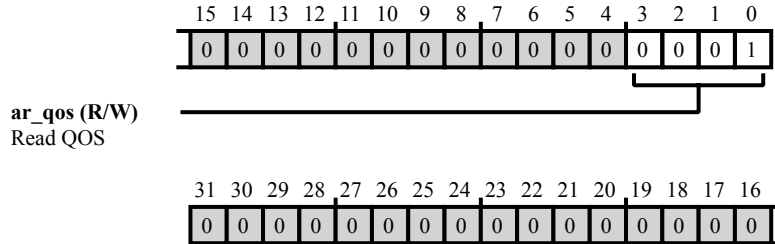


Figure 41-18: SCB0_CRC1_CH1_READ_QOS Register Diagram

Table 41-20: SCB0_CRC1_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

CRC1 Channel 1 Write Quality of Service Register

The `SCB0_CRC1_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

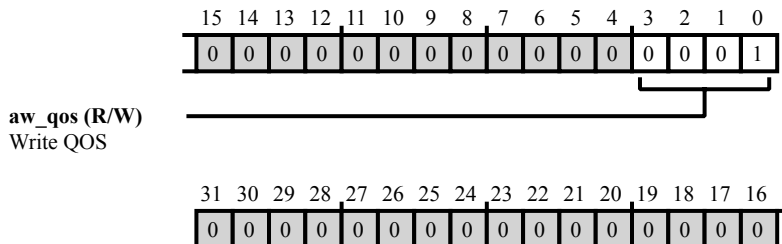


Figure 41-19: SCB0_CRC1_CH1_WRITE_QOS Register Diagram

Table 41-21: SCB0_CRC1_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

Crypto Read Quality of Service Register

The `SCB0_CRYPT0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

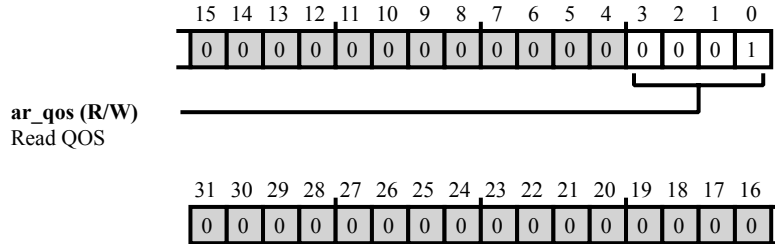


Figure 41-20: SCB0_CRYPT0_READ_QOS Register Diagram

Table 41-22: SCB0_CRYPT0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

CRYPTO Write Quality of Service Register

The `SCB0_CRYPT0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

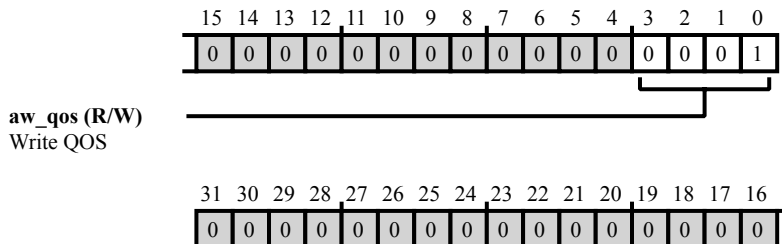


Figure 41-21: SCB0_CRYPT0_WRITE_QOS Register Diagram

Table 41-23: SCB0_CRYPT0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

DBG Read Quality of Service Register

The `SCB0_DBG_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

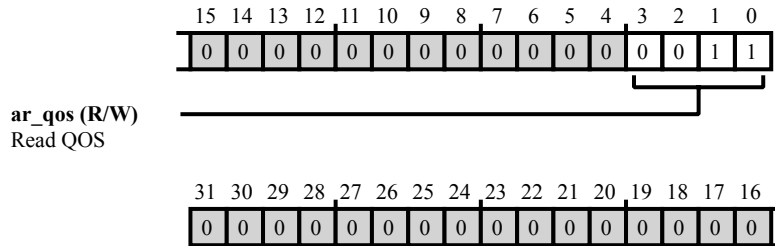


Figure 41-22: SCB0_DBG_READ_QOS Register Diagram

Table 41-24: SCB0_DBG_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

DBG Write Quality of Service Register

The `SCB0_DBG_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

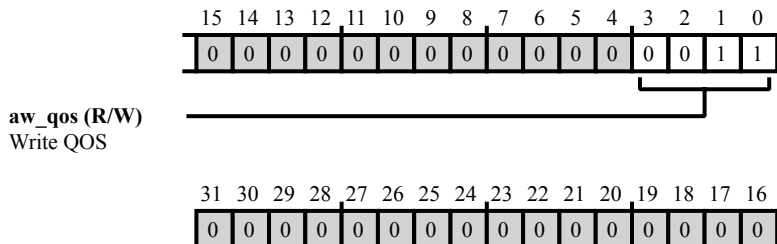


Figure 41-23: SCB0_DBG_WRITE_QOS Register Diagram

Table 41-25: SCB0_DBG_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

DLDMA0 Channel 0 Read Quality of Service Register

The `SCB0_DLDMA0_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

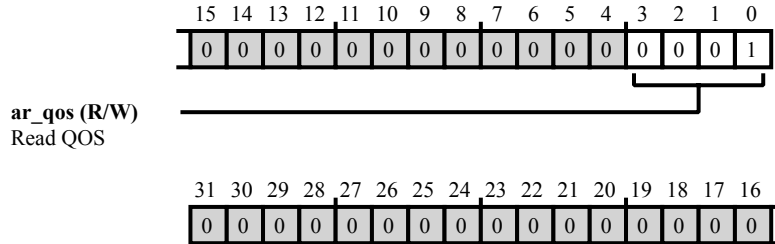


Figure 41-24: SCB0_DLDMA0_CH0_READ_QOS Register Diagram

Table 41-26: SCB0_DLDMA0_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

DLDMA0 Channel 0 Write Quality of Service Register

The `SCB0_DLDMA0_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

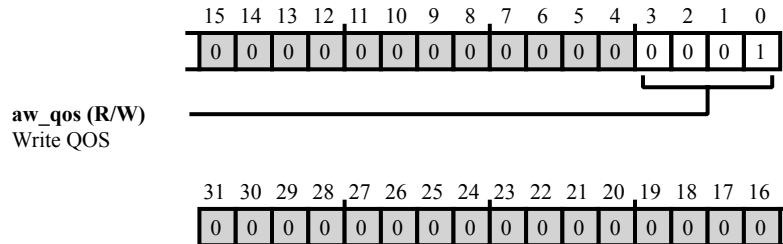


Figure 41-25: SCB0_DLDMA0_CH0_WRITE_QOS Register Diagram

Table 41-27: SCB0_DLDMA0_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

DLDMA0 Channel 1 Read Quality of Service Register

The `SCB0_DLDMA0_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

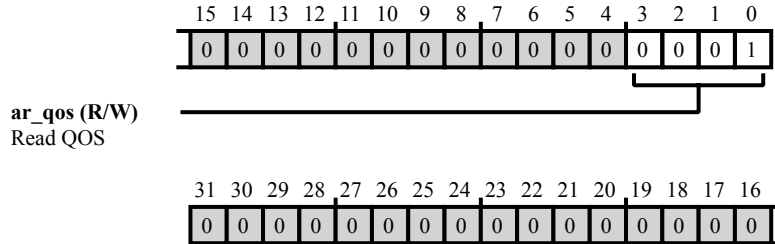


Figure 41-26: SCB0_DLDMA0_CH1_READ_QOS Register Diagram

Table 41-28: SCB0_DLDMA0_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

DLDMA0 Channel 1 Write Quality of Service Register

The `SCB0_DLDMA0_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

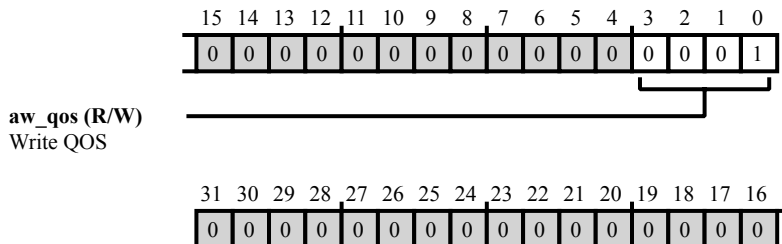


Figure 41-27: SCB0_DLDMA0_CH1_WRITE_QOS Register Diagram

Table 41-29: SCB0_DLDMA0_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

DLDMA1 Channel 0 Read Quality of Service Register

The `SCB0_DLDMA1_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

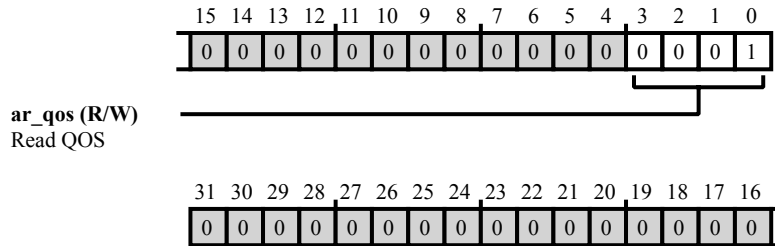


Figure 41-28: SCB0_DLDMA1_CH0_READ_QOS Register Diagram

Table 41-30: SCB0_DLDMA1_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

DLDMA1 Channel 0 Write Quality of Service Register

The `SCB0_DLDMA1_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

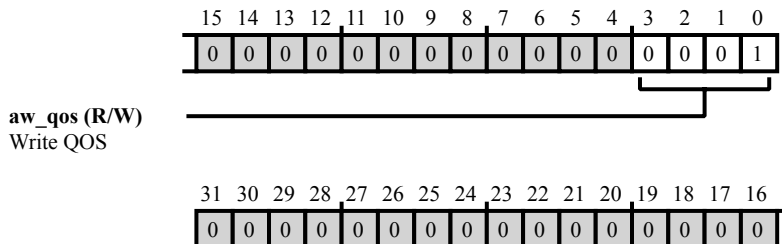


Figure 41-29: SCB0_DLDMA1_CH0_WRITE_QOS Register Diagram

Table 41-31: SCB0_DLDMA1_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

DLDMA1 Channel 1 Read Quality of Service Register

The `SCB0_DLDMA1_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

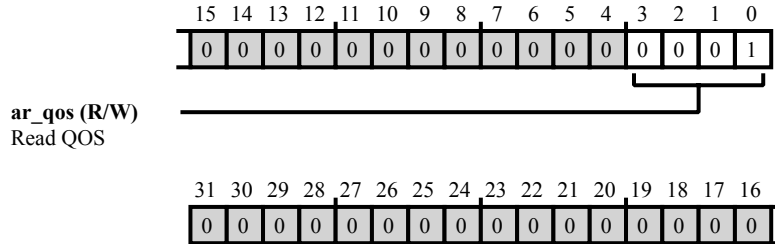


Figure 41-30: SCB0_DLDMA1_CH1_READ_QOS Register Diagram

Table 41-32: SCB0_DLDMA1_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

DLDMA1 Channel 1 Write Quality of Service Register

The `SCB0_DLDMA1_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

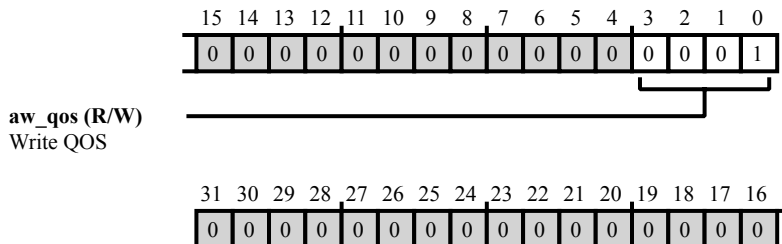


Figure 41-31: SCB0_DLDMA1_CH1_WRITE_QOS Register Diagram

Table 41-33: SCB0_DLDMA1_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

ETR Read Quality of Service Register

The `SCB0_ETR_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

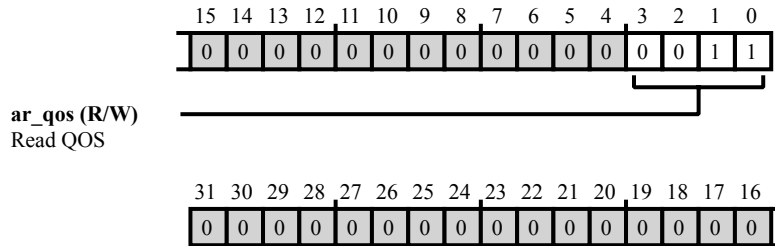


Figure 41-32: SCB0_ETR_READ_QOS Register Diagram

Table 41-34: SCB0_ETR_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

ETR Write Quality of Service Register

The `SCB0_ETR_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

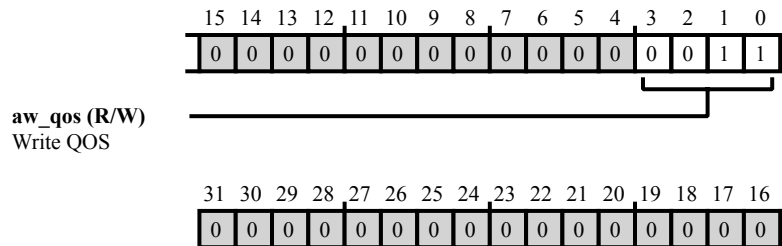


Figure 41-33: `SCB0_ETR_WRITE_QOS` Register Diagram

Table 41-35: `SCB0_ETR_WRITE_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	<code>AW_QOS</code>	Write QOS.

HSMDMA Channel 0 Read Quality of Service Register

The `SCB0_HSMDMA_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

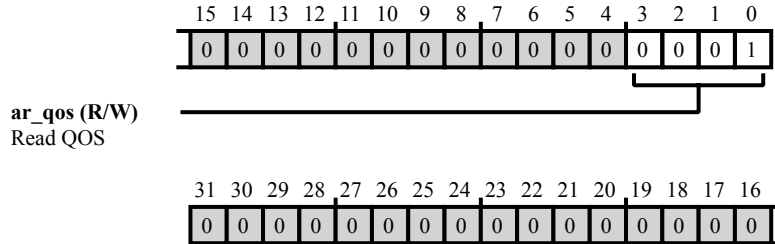


Figure 41-34: SCB0_HSMDMA_CH0_READ_QOS Register Diagram

Table 41-36: SCB0_HSMDMA_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

HSMDMA Channel 0 Write Quality of Service Register

The `SCB0_HSMDMA_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

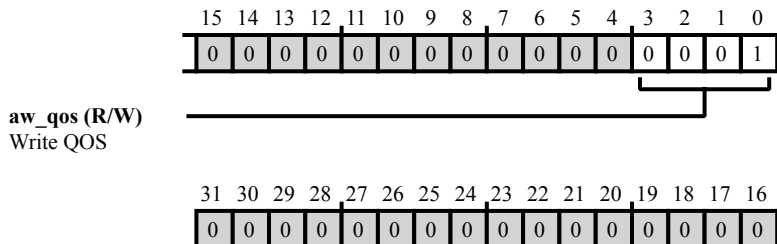


Figure 41-35: SCB0_HSMDMA_CH0_WRITE_QOS Register Diagram

Table 41-37: SCB0_HSMDMA_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

HSMDMA Channel 1 Read Quality of Service Register

The `SCB0_HSMDMA_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

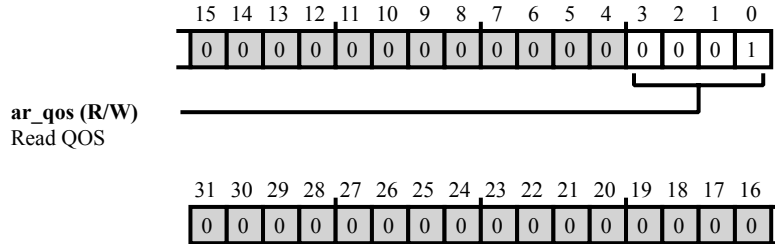


Figure 41-36: SCB0_HSMDMA_CH1_READ_QOS Register Diagram

Table 41-38: SCB0_HSMDMA_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

HSMDMA Channel 1 Write Quality of Service Register

The `SCB0_HSMDMA_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

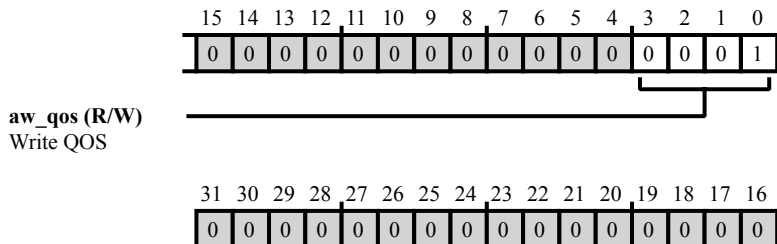


Figure 41-37: SCB0_HSMDMA_CH1_WRITE_QOS Register Diagram

Table 41-39: SCB0_HSMDMA_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

LPO Read Quality of Service Register

The `SCB0_LP0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

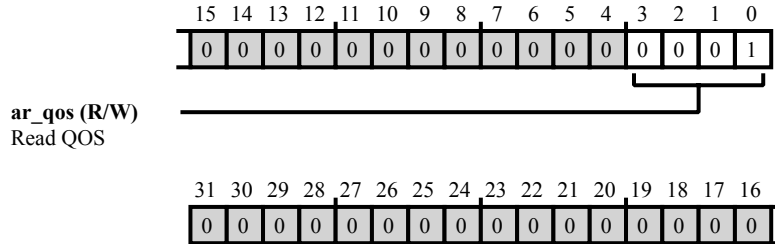


Figure 41-38: SCB0_LP0_READ_QOS Register Diagram

Table 41-40: SCB0_LP0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

LPO Write Quality of Service Register

The `SCB0_LP0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

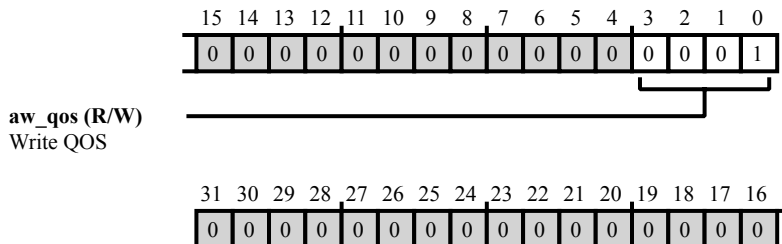


Figure 41-39: SCB0_LP0_WRITE_QOS Register Diagram

Table 41-41: SCB0_LP0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

LP1 Read Quality of Service Register

The `SCB0_LP1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

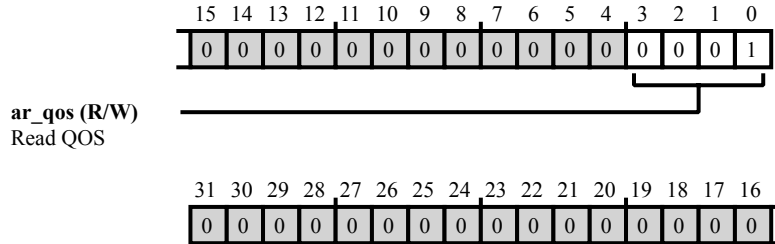


Figure 41-40: SCB0_LP1_READ_QOS Register Diagram

Table 41-42: SCB0_LP1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

LP1 Write Quality of Service Register

The `SCB0_LP1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

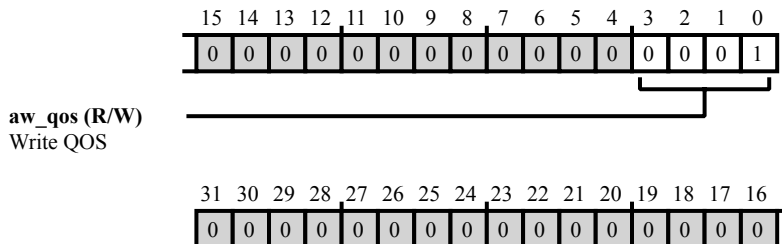


Figure 41-41: SCB0_LP1_WRITE_QOS Register Diagram

Table 41-43: SCB0_LP1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

MLB Read Quality of Service Register

The `SCB0_MLB_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

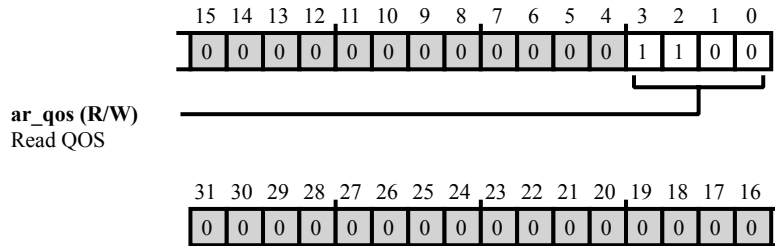


Figure 41-42: `SCB0_MLB_READ_QOS` Register Diagram

Table 41-44: `SCB0_MLB_READ_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

MLB Write Quality of Service Register

The `SCB0_MLB_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

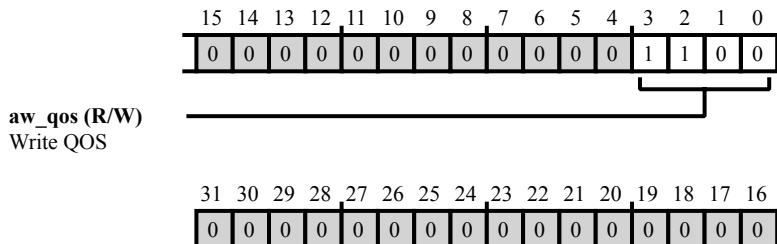


Figure 41-43: SCB0_MLB_WRITE_QOS Register Diagram

Table 41-45: SCB0_MLB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

MSMDMA Channel 0 Read Quality of Service Register

The `SCB0_MSMDMA_CH0_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

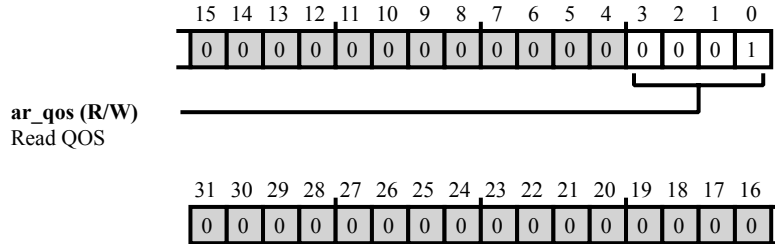


Figure 41-44: SCB0_MSMDMA_CH0_READ_QOS Register Diagram

Table 41-46: SCB0_MSMDMA_CH0_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

MSMDMA Channel 0 Write Quality of Service Register

The `SCB0_MSMDMA_CH0_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

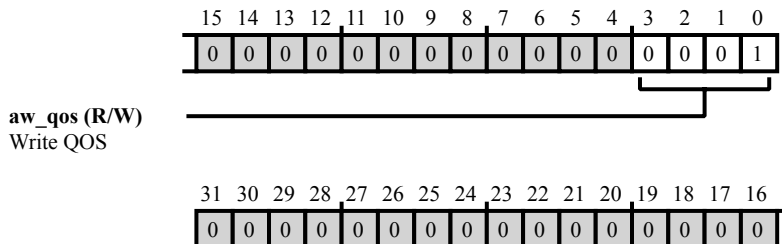


Figure 41-45: SCB0_MSMDMA_CH0_WRITE_QOS Register Diagram

Table 41-47: SCB0_MSMDMA_CH0_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

MSMDMA Channel 1 Read Quality of Service Register

The `SCB0_MSMDMA_CH1_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

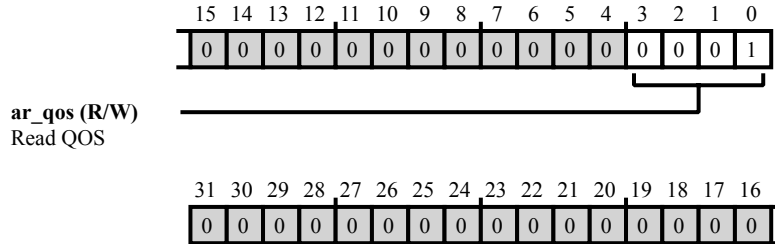


Figure 41-46: SCB0_MSMDMA_CH1_READ_QOS Register Diagram

Table 41-48: SCB0_MSMDMA_CH1_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

MSMDMA Channel 1 Write Quality of Service Register

The `SCB0_MSMDMA_CH1_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

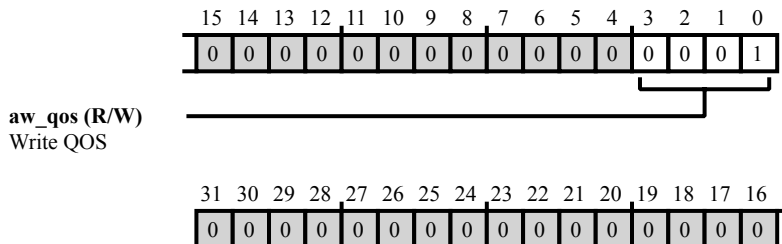


Figure 41-47: SCB0_MSMDMA_CH1_WRITE_QOS Register Diagram

Table 41-49: SCB0_MSMDMA_CH1_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SH0 DPORT Read Quality of Service Register

The `SCB0_SH0_DPORT_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

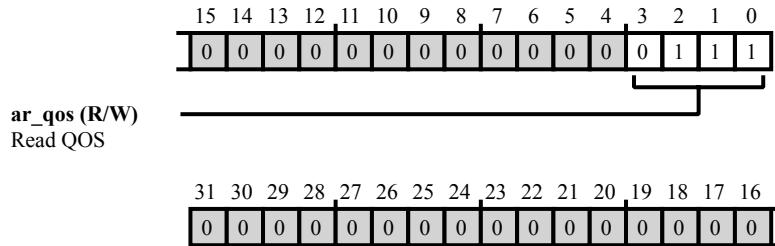


Figure 41-48: SCB0_SH0_DPORT_READ_QOS Register Diagram

Table 41-50: SCB0_SH0_DPORT_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SH0 DPORT Write Quality of Service Register

The `SCB0_SH0_DPORT_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

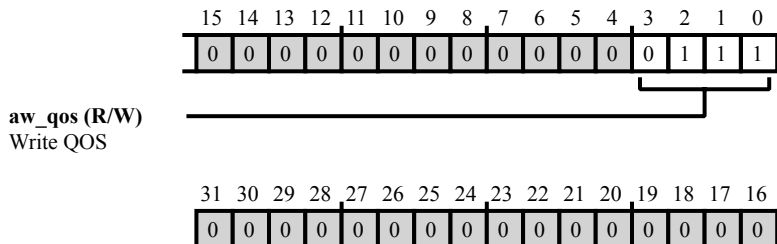


Figure 41-49: SCB0_SH0_DPORT_WRITE_QOS Register Diagram

Table 41-51: SCB0_SH0_DPORT_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SH0 IPORT Read Quality of Service Register

The `SCB0_SH0_IPORT_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

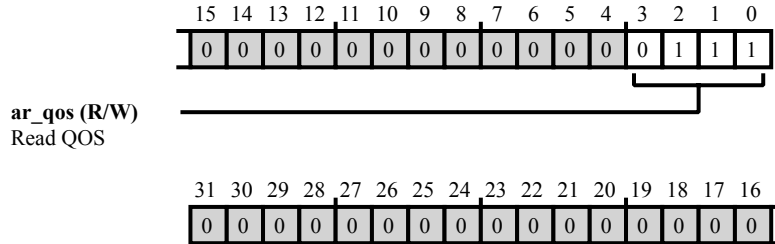


Figure 41-50: SCB0_SH0_IPORT_READ_QOS Register Diagram

Table 41-52: SCB0_SH0_IPORT_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SH0 IPORT Write Quality of Service Register

The `SCB0_SH0_IPORT_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

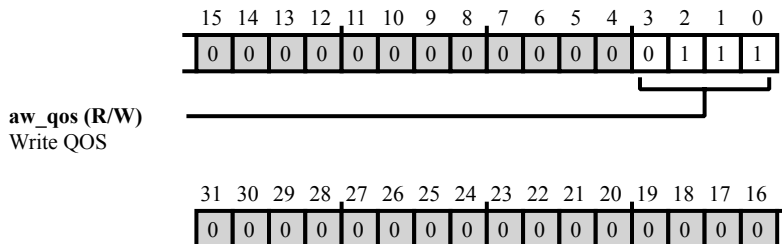


Figure 41-51: SCB0_SH0_IPORT_WRITE_QOS Register Diagram

Table 41-53: SCB0_SH0_IPORT_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SH0 MMR Read Quality of Service Register

The `SCB0_SH0_MMR_IB_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

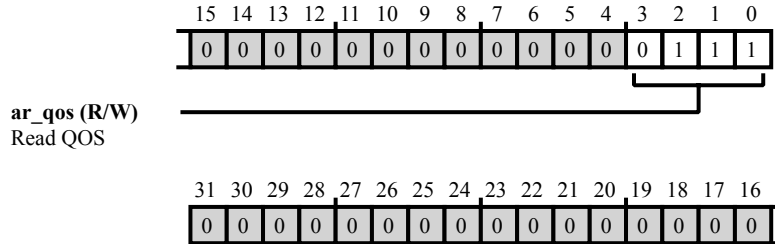


Figure 41-52: SCB0_SH0_MMR_IB_READ_QOS Register Diagram

Table 41-54: SCB0_SH0_MMR_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SH0 MMR Write Quality of Service Register

The `SCB0_SH0_MMR_IB_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

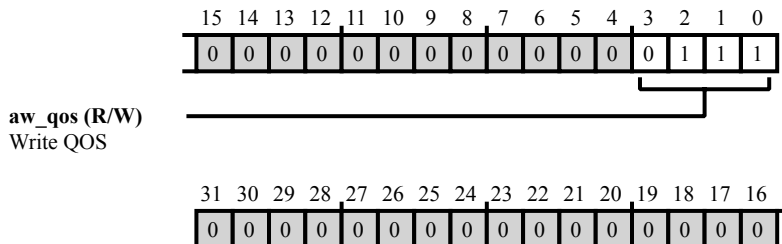


Figure 41-53: SCB0_SH0_MMR_IB_WRITE_QOS Register Diagram

Table 41-55: SCB0_SH0_MMR_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP0A Read Quality of Service Register

The `SCB0_SP0A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

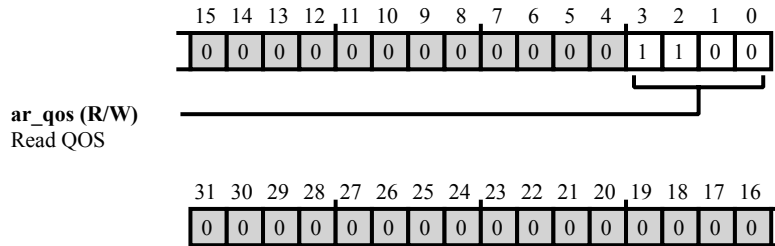


Figure 41-54: SCB0_SP0A_READ_QOS Register Diagram

Table 41-56: SCB0_SP0A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP0A Write Quality of Service Register

The `SCB0_SP0A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

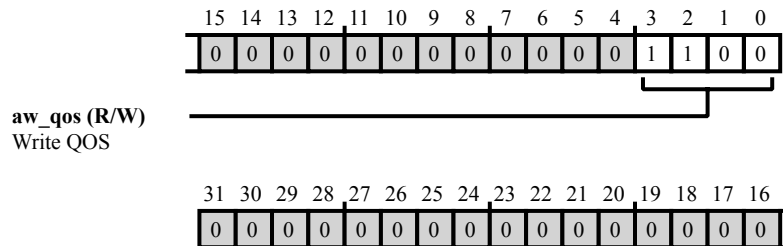


Figure 41-55: `SCB0_SP0A_WRITE_QOS` Register Diagram

Table 41-57: `SCB0_SP0A_WRITE_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	<code>AW_QOS</code>	Write QOS.

SPOB Read Quality of Service Register

The `SCB0_SP0B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

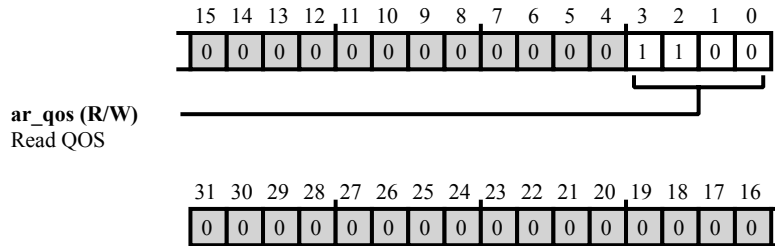


Figure 41-56: SCB0_SP0B_READ_QOS Register Diagram

Table 41-58: SCB0_SP0B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPOB Write Quality of Service Register

The `SCB0_SP0B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

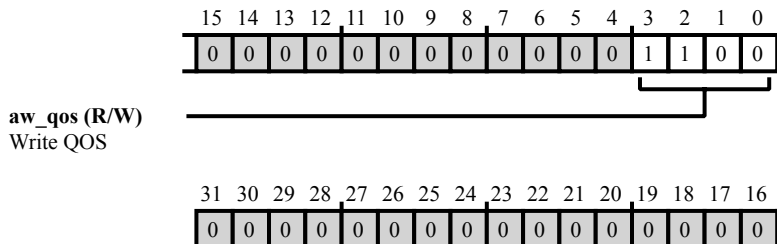


Figure 41-57: SCB0_SP0B_WRITE_QOS Register Diagram

Table 41-59: SCB0_SP0B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP1A Read Quality of Service Register

The `SCB0_SP1A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

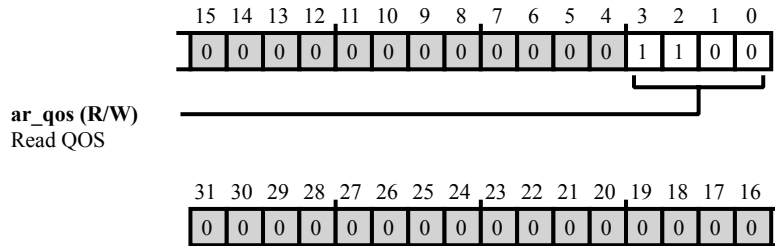


Figure 41-58: SCB0_SP1A_READ_QOS Register Diagram

Table 41-60: SCB0_SP1A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP1A Write Quality of Service Register

The `SCB0_SP1A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

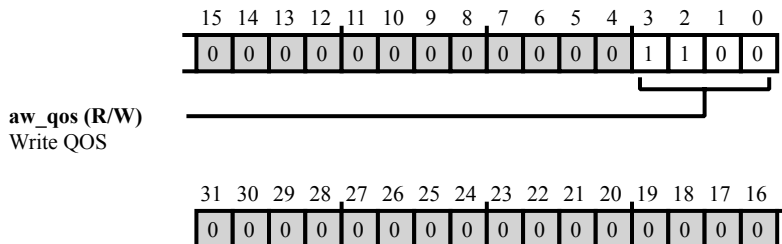


Figure 41-59: SCB0_SP1A_WRITE_QOS Register Diagram

Table 41-61: SCB0_SP1A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP1B Read Quality of Service Register

The `SCB0_SP1B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

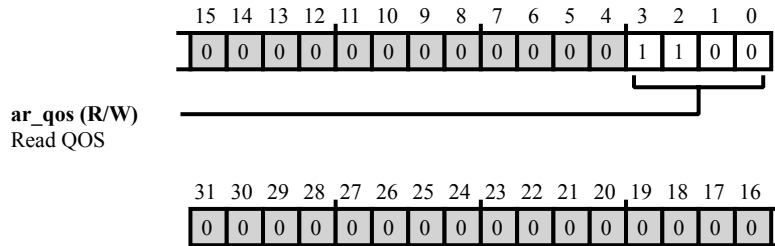


Figure 41-60: SCB0_SP1B_READ_QOS Register Diagram

Table 41-62: SCB0_SP1B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP1B Write Quality of Service Register

The `SCB0_SP1B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

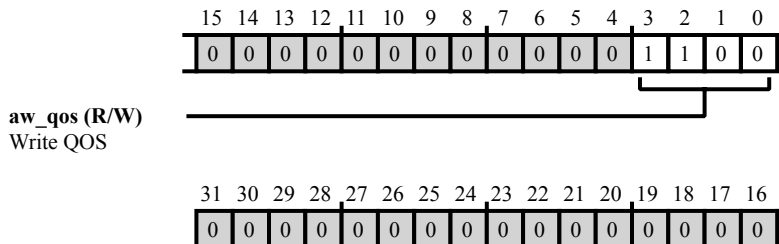


Figure 41-61: SCB0_SP1B_WRITE_QOS Register Diagram

Table 41-63: SCB0_SP1B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP2A Read Quality of Service Register

The `SCB0_SP2A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

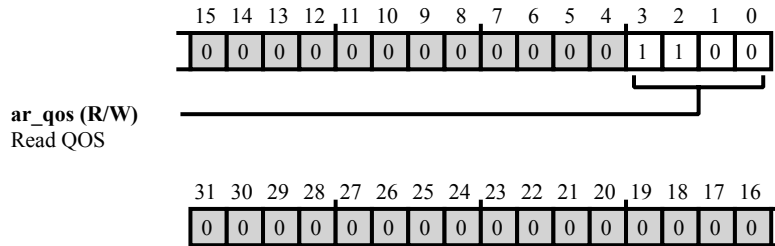


Figure 41-62: SCB0_SP2A_READ_QOS Register Diagram

Table 41-64: SCB0_SP2A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP2A Write Quality of Service Register

The `SCB0_SP2A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

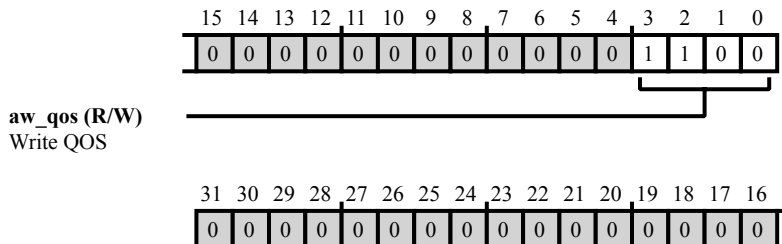


Figure 41-63: SCB0_SP2A_WRITE_QOS Register Diagram

Table 41-65: SCB0_SP2A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP2B Read Quality of Service Register

The `SCB0_SP2B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

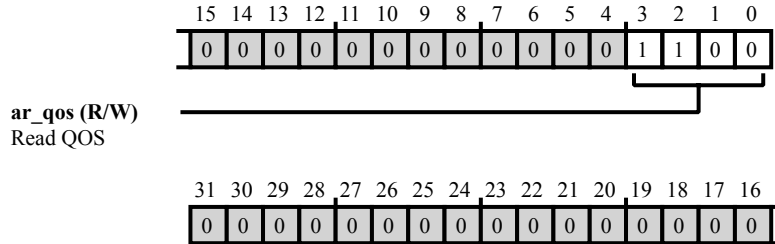


Figure 41-64: SCB0_SP2B_READ_QOS Register Diagram

Table 41-66: SCB0_SP2B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP2B Write Quality of Service Register

The `SCB0_SP2B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

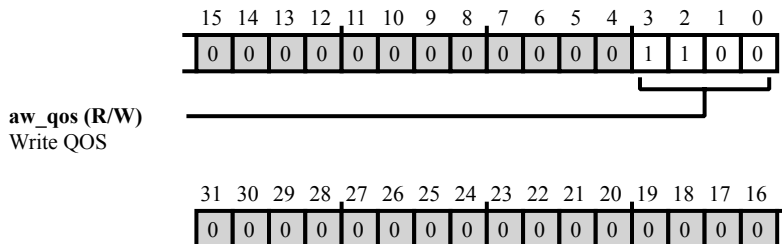


Figure 41-65: SCB0_SP2B_WRITE_QOS Register Diagram

Table 41-67: SCB0_SP2B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP3A Read Quality of Service Register

The `SCB0_SP3A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

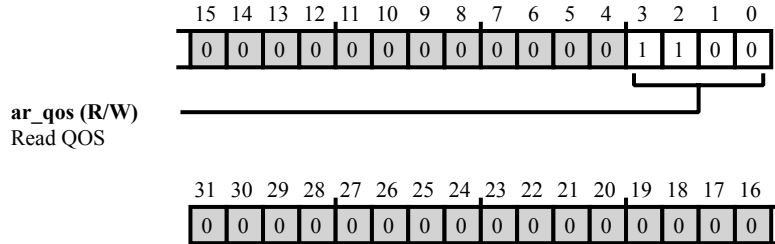


Figure 41-66: SCB0_SP3A_READ_QOS Register Diagram

Table 41-68: SCB0_SP3A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP3A Write Quality of Service Register

The `SCB0_SP3A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

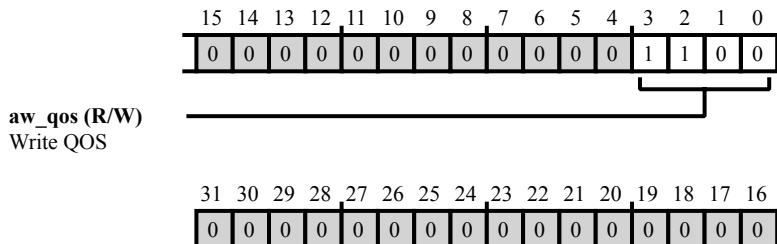


Figure 41-67: SCB0_SP3A_WRITE_QOS Register Diagram

Table 41-69: SCB0_SP3A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP3B Read Quality of Service Register

The `SCB0_SP3B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

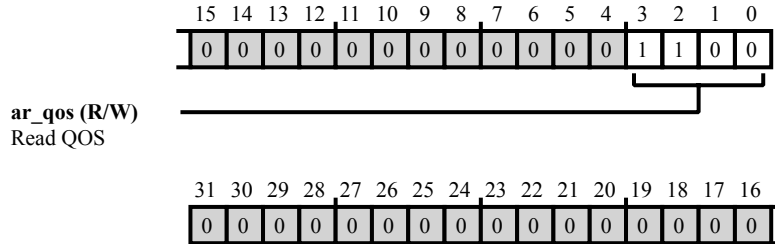


Figure 41-68: SCB0_SP3B_READ_QOS Register Diagram

Table 41-70: SCB0_SP3B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP3B Write Quality of Service Register

The `SCB0_SP3B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

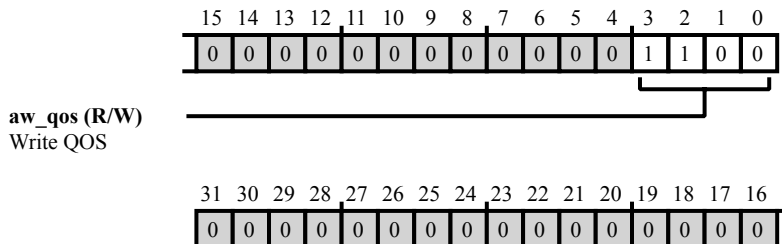


Figure 41-69: SCB0_SP3B_WRITE_QOS Register Diagram

Table 41-71: SCB0_SP3B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP4A Read Quality of Service Register

The `SCB0_SP4A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

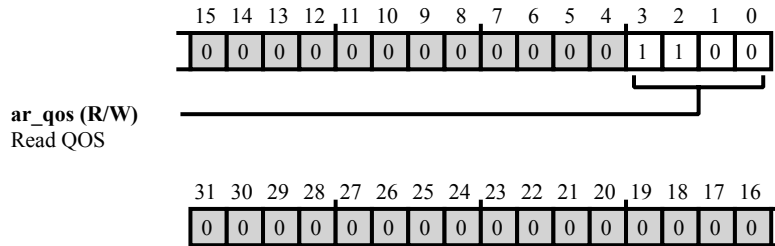


Figure 41-70: SCB0_SP4A_READ_QOS Register Diagram

Table 41-72: SCB0_SP4A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP4A Write Quality of Service Register

The `SCB0_SP4A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

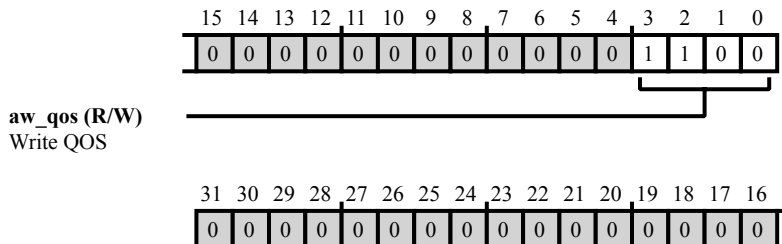


Figure 41-71: SCB0_SP4A_WRITE_QOS Register Diagram

Table 41-73: SCB0_SP4A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP4B Read Quality of Service Register

The `SCB0_SP4B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

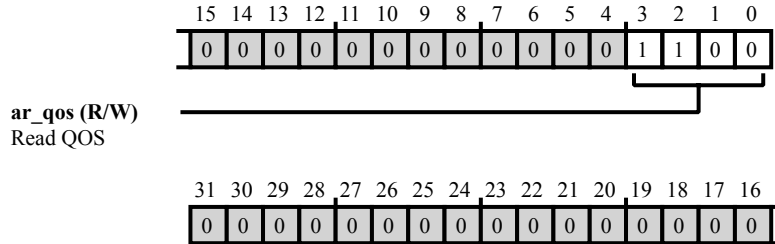


Figure 41-72: SCB0_SP4B_READ_QOS Register Diagram

Table 41-74: SCB0_SP4B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP4B Write Quality of Service Register

The `SCB0_SP4B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

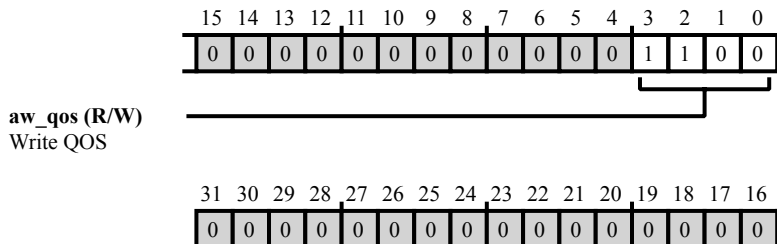


Figure 41-73: SCB0_SP4B_WRITE_QOS Register Diagram

Table 41-75: SCB0_SP4B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP5A Read Quality of Service Register

The `SCB0_SP5A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

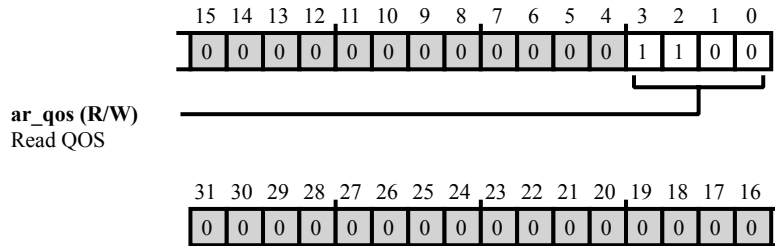


Figure 41-74: SCB0_SP5A_READ_QOS Register Diagram

Table 41-76: SCB0_SP5A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP5A Write Quality of Service Register

The `SCB0_SP5A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

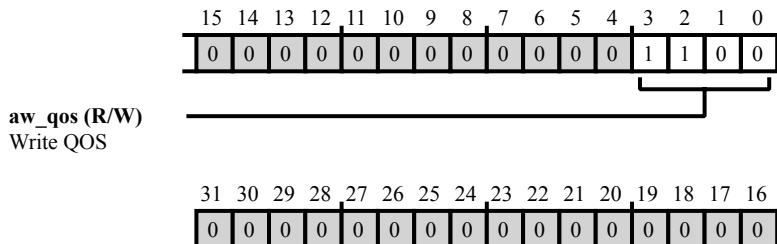


Figure 41-75: SCB0_SP5A_WRITE_QOS Register Diagram

Table 41-77: SCB0_SP5A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP5B Read Quality of Service Register

The `SCB0_SP5B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

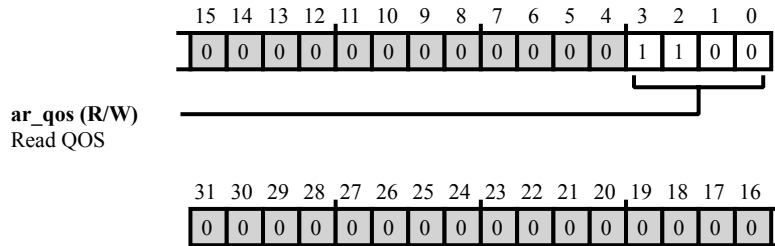


Figure 41-76: SCB0_SP5B_READ_QOS Register Diagram

Table 41-78: SCB0_SP5B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP5B Write Quality of Service Register

The `SCB0_SP5B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

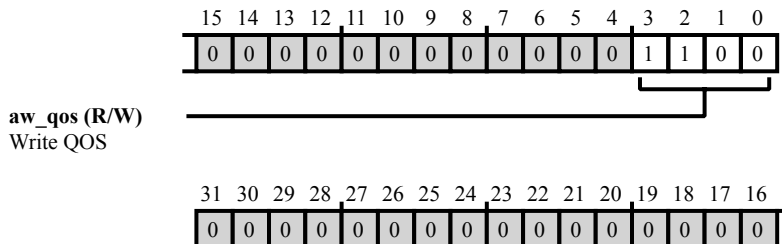


Figure 41-77: SCB0_SP5B_WRITE_QOS Register Diagram

Table 41-79: SCB0_SP5B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP6A Read Quality of Service Register

The `SCB0_SP6A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

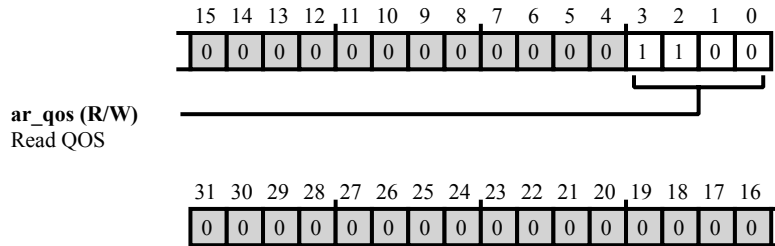


Figure 41-78: SCB0_SP6A_READ_QOS Register Diagram

Table 41-80: SCB0_SP6A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP6A Write Quality of Service Register

The `SCB0_SP6A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

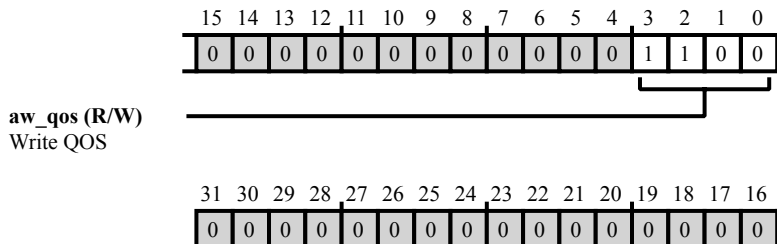


Figure 41-79: SCB0_SP6A_WRITE_QOS Register Diagram

Table 41-81: SCB0_SP6A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP6B Read Quality of Service Register

The `SCB0_SP6B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

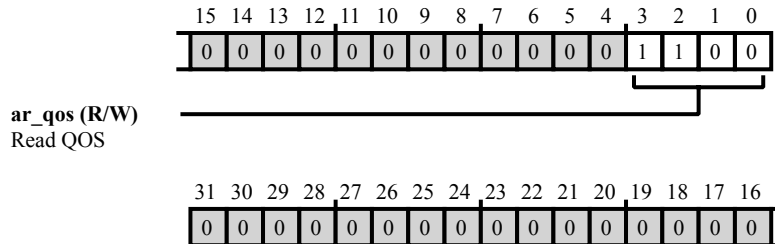


Figure 41-80: `SCB0_SP6B_READ_QOS` Register Diagram

Table 41-82: `SCB0_SP6B_READ_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP6B Write Quality of Service Register

The `SCB0_SP6B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

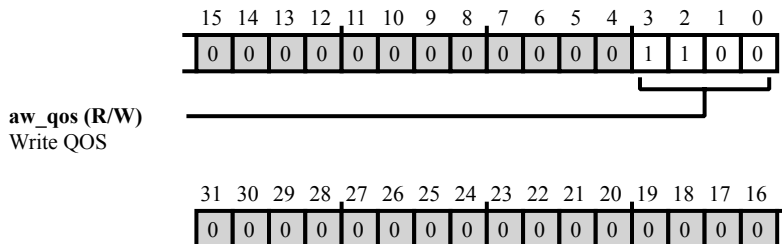


Figure 41-81: SCB0_SP6B_WRITE_QOS Register Diagram

Table 41-83: SCB0_SP6B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP7A Read Quality of Service Register

The `SCB0_SP7A_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

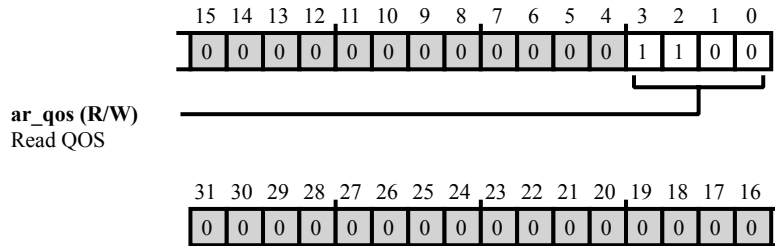


Figure 41-82: SCB0_SP7A_READ_QOS Register Diagram

Table 41-84: SCB0_SP7A_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP7A Write Quality of Service Register

The `SCB0_SP7A_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

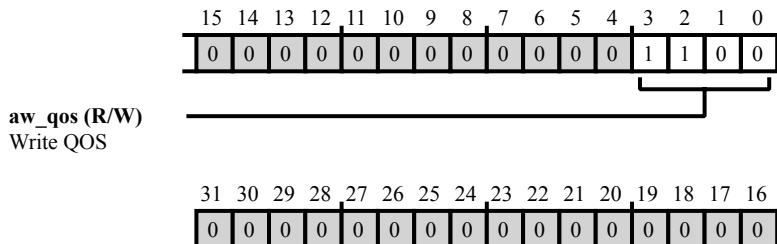


Figure 41-83: SCB0_SP7A_WRITE_QOS Register Diagram

Table 41-85: SCB0_SP7A_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SP7B Read Quality of Service Register

The `SCB0_SP7B_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

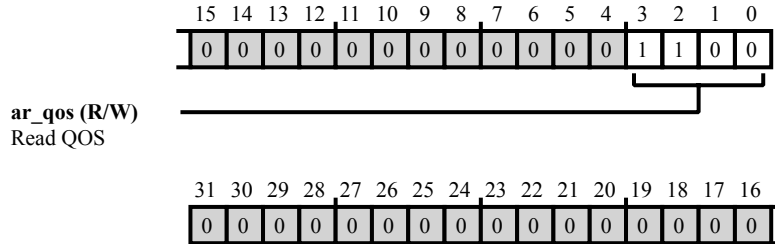


Figure 41-84: SCB0_SP7B_READ_QOS Register Diagram

Table 41-86: SCB0_SP7B_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SP7B Write Quality of Service Register

The `SCB0_SP7B_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

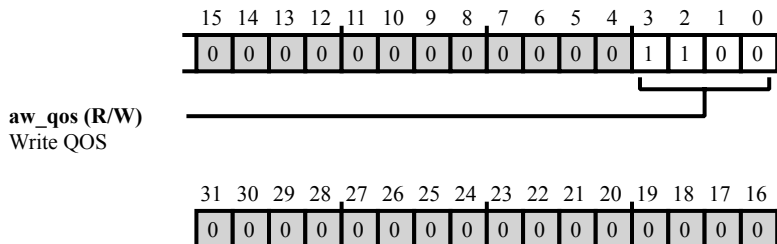


Figure 41-85: SCB0_SP7B_WRITE_QOS Register Diagram

Table 41-87: SCB0_SP7B_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SPI0 RX Read Quality of Service Register

The `SCB0_SPI0RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

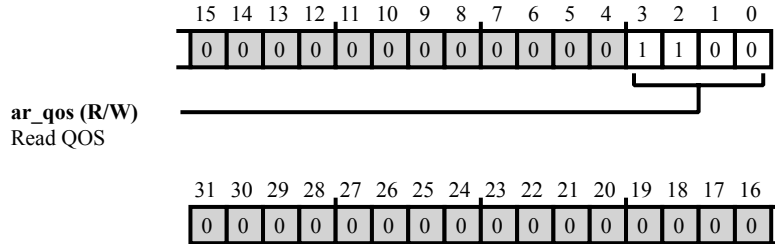


Figure 41-86: SCB0_SPI0RX_READ_QOS Register Diagram

Table 41-88: SCB0_SPI0RX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI0 RX Write Quality of Service Register

The `SCB0_SPI0RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

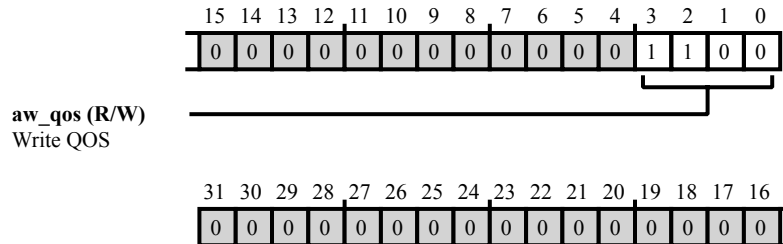


Figure 41-87: SCB0_SPI0RX_WRITE_QOS Register Diagram

Table 41-89: SCB0_SPI0RX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SPI0 TX Read Quality of Service Register

The `SCB0_SPI0TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

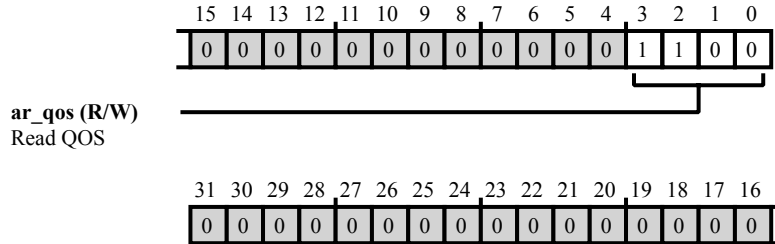


Figure 41-88: SCB0_SPI0TX_READ_QOS Register Diagram

Table 41-90: SCB0_SPI0TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI0 TX Write Quality of Service Register

The `SCB0_SPI0TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

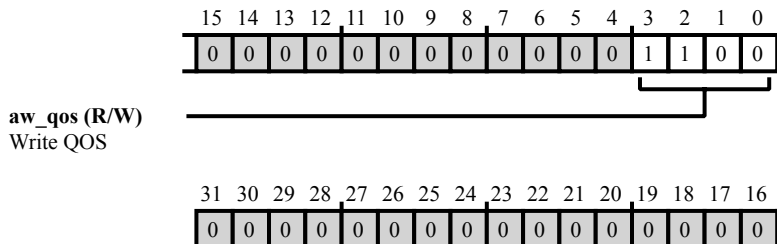


Figure 41-89: SCB0_SPI0TX_WRITE_QOS Register Diagram

Table 41-91: SCB0_SPI0TX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SPI1 RX Read Quality of Service Register

The `SCB0_SPI1RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

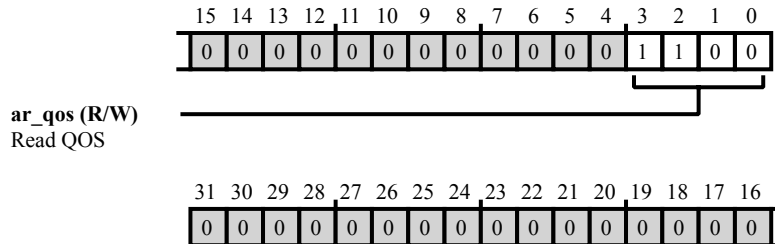


Figure 41-90: `SCB0_SPI1RX_READ_QOS` Register Diagram

Table 41-92: `SCB0_SPI1RX_READ_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI1 RX Write Quality of Service Register

The `SCB0_SPI1RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

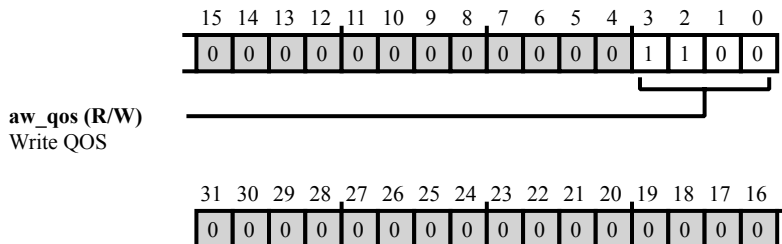


Figure 41-91: SCB0_SPI1RX_WRITE_QOS Register Diagram

Table 41-93: SCB0_SPI1RX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SPI1 TX Read Quality of Service Register

The `SCB0_SPI1TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

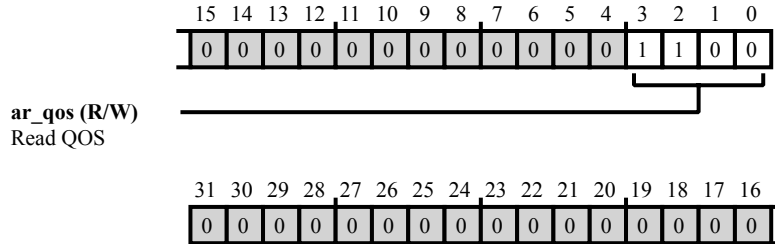


Figure 41-92: SCB0_SPI1TX_READ_QOS Register Diagram

Table 41-94: SCB0_SPI1TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI1 TX Write Quality of Service Register

The `SCB0_SPI1TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

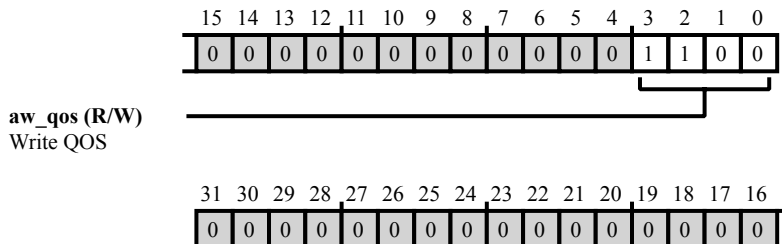


Figure 41-93: SCB0_SPI1TX_WRITE_QOS Register Diagram

Table 41-95: SCB0_SPI1TX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

SPI2 RX Read Quality of Service Register

The `SCB0_SPI2RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

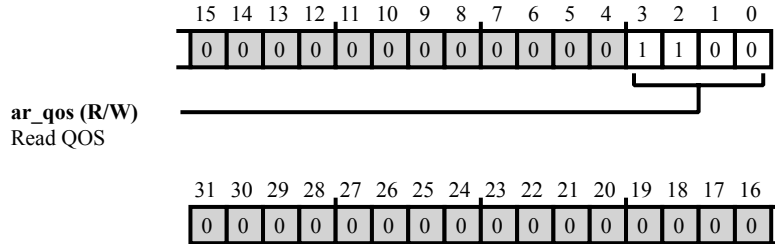


Figure 41-94: SCB0_SPI2RX_READ_QOS Register Diagram

Table 41-96: SCB0_SPI2RX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI2 RX Write Quality of Service Register

The `SCB0_SPI2RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

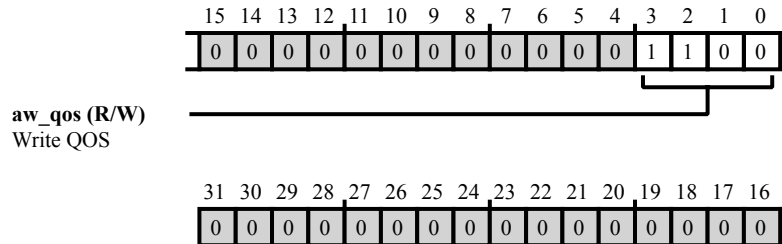


Figure 41-95: `SCB0_SPI2RX_WRITE_QOS` Register Diagram

Table 41-97: `SCB0_SPI2RX_WRITE_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	<code>AW_QOS</code>	Write QOS.

SPI2 TX Read Quality of Service Register

The `SCB0_SPI2TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

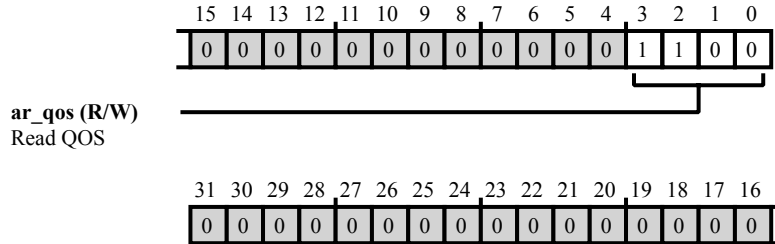


Figure 41-96: SCB0_SPI2TX_READ_QOS Register Diagram

Table 41-98: SCB0_SPI2TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

SPI2 TX Write Quality of Service Register

The `SCB0_SPI2TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

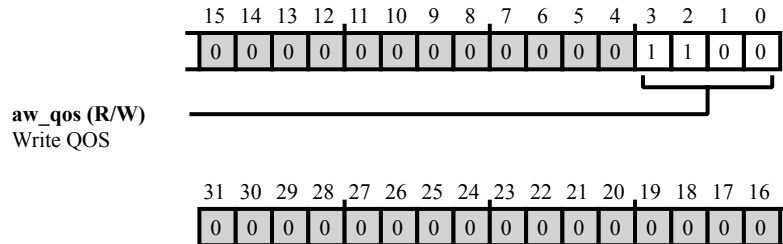


Figure 41-97: `SCB0_SPI2TX_WRITE_QOS` Register Diagram

Table 41-99: `SCB0_SPI2TX_WRITE_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	<code>AW_QOS</code>	Write QOS.

UART0 RX Read Quality of Service Register

The `SCB0_UART0_RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

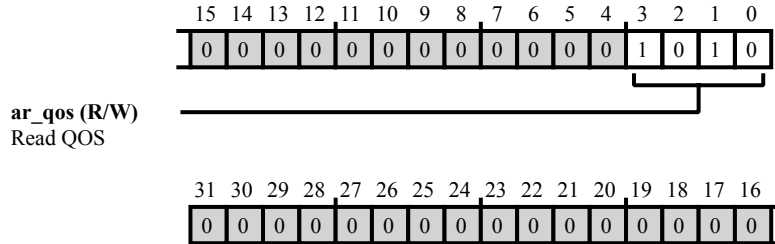


Figure 41-98: SCB0_UART0_RX_READ_QOS Register Diagram

Table 41-100: SCB0_UART0_RX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART0 RX Write Quality of Service Register

The `SCB0_UART0_RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

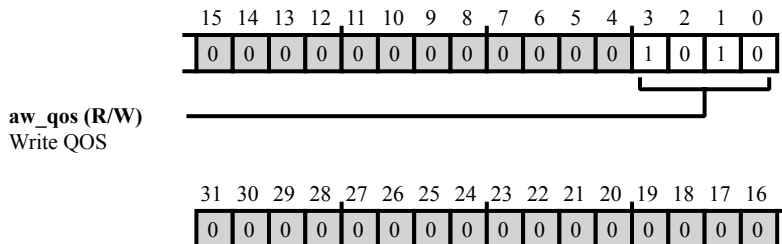


Figure 41-99: SCB0_UART0_RX_WRITE_QOS Register Diagram

Table 41-101: SCB0_UART0_RX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

UART0 TX Read Quality of Service Register

The `SCB0_UART0_TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

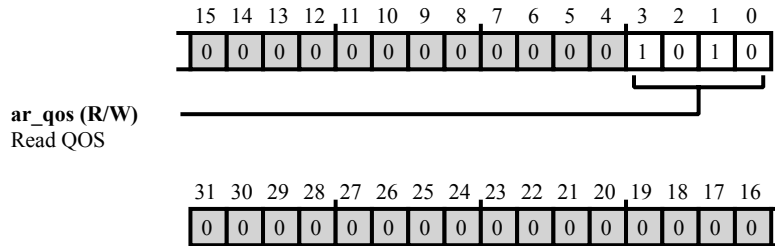


Figure 41-100: SCB0_UART0_TX_READ_QOS Register Diagram

Table 41-102: SCB0_UART0_TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART0 TX Write Quality of Service Register

The `SCB0_UART0_TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

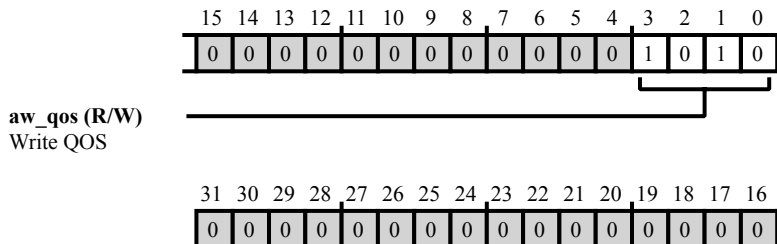


Figure 41-101: SCB0_UART0_TX_WRITE_QOS Register Diagram

Table 41-103: SCB0_UART0_TX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

UART1 TX Read Quality of Service Register

The `SCB0_UART1_RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

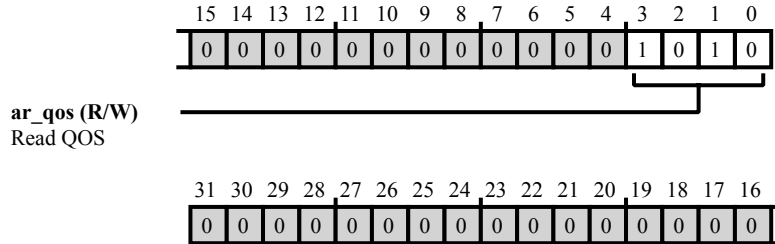


Figure 41-102: SCB0_UART1_RX_READ_QOS Register Diagram

Table 41-104: SCB0_UART1_RX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART1 RX Write Quality of Service Register

The `SCB0_UART1_RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

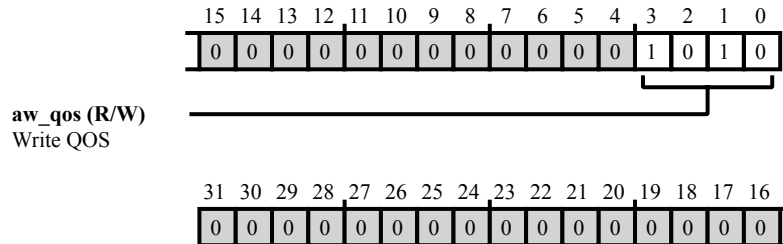


Figure 41-103: `SCB0_UART1_RX_WRITE_QOS` Register Diagram

Table 41-105: `SCB0_UART1_RX_WRITE_QOS` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	<code>AW_QOS</code>	Write QOS.

UART1 TX Read Quality of Service Register

The `SCB0_UART1_TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

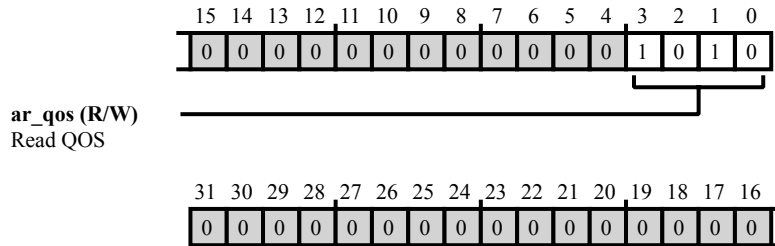


Figure 41-104: SCB0_UART1_TX_READ_QOS Register Diagram

Table 41-106: SCB0_UART1_TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART1 TX Write Quality of Service Register

The `SCB0_UART1_TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

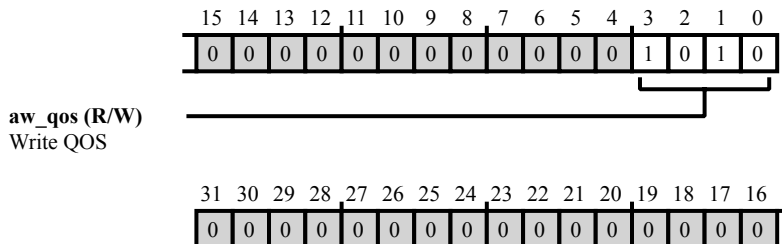


Figure 41-105: SCB0_UART1_TX_WRITE_QOS Register Diagram

Table 41-107: SCB0_UART1_TX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

UART2 RX Read Quality of Service Register

The `SCB0_UART2_RX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

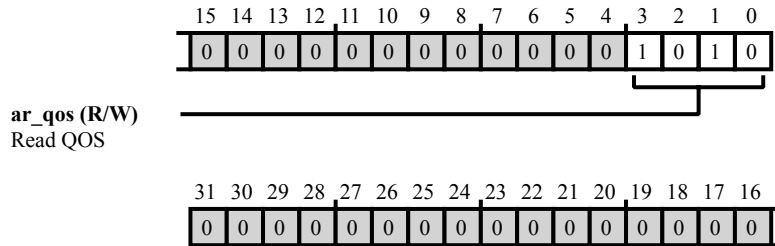


Figure 41-106: SCB0_UART2_RX_READ_QOS Register Diagram

Table 41-108: SCB0_UART2_RX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART2 RX Write Quality of Service Register

The `SCB0_UART2_RX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

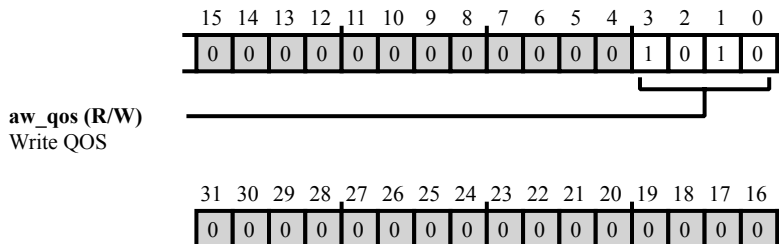


Figure 41-107: SCB0_UART2_RX_WRITE_QOS Register Diagram

Table 41-109: SCB0_UART2_RX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

UART2 TX Read Quality of Service Register

The `SCB0_UART2_TX_READ_QOS` register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting read channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

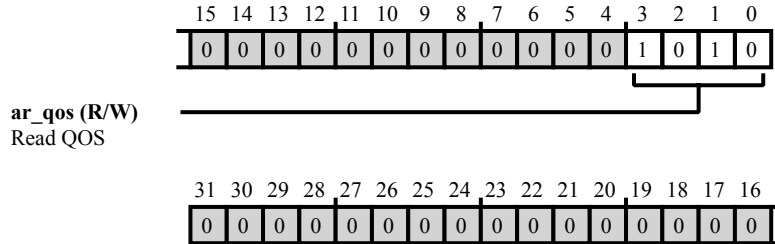


Figure 41-108: SCB0_UART2_TX_READ_QOS Register Diagram

Table 41-110: SCB0_UART2_TX_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Read QOS.

UART2 TX Write Quality of Service Register

The `SCB0_UART2_TX_WRITE_QOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among the masters requesting write channel access. For the mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

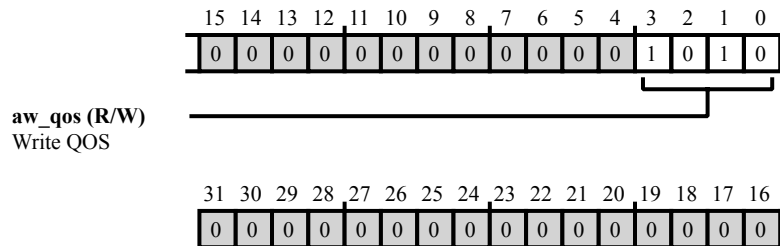


Figure 41-109: SCB0_UART2_TX_WRITE_QOS Register Diagram

Table 41-111: SCB0_UART2_TX_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Write QOS.

ADSP-2156x SCB1 Register Descriptions

System Crossbar for DMC Memory Space (SCB1) contains the following registers.

Table 41-112: ADSP-2156x SCB1 Register List

Name	Description
<code>SCB1_MST_IB_SYNC_MODE</code>	Master Synchronization Mode Register

Master Synchronization Mode Register

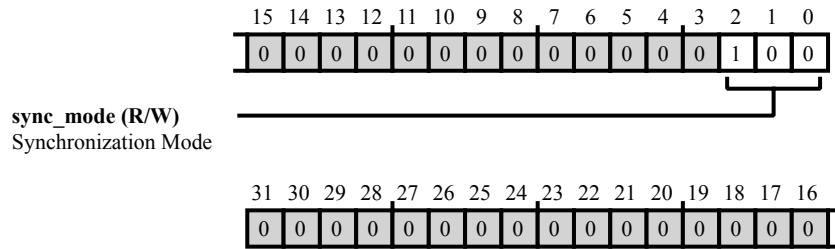


Figure 41-110: SCB1_MST_IB_SYNC_MODE Register Diagram

Table 41-113: SCB1_MST_IB_SYNC_MODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	SYNC_MODE	Synchronization Mode. The <code>SCB1_MST_IB_SYNC_MODE</code> bit field configures the SYSCLK-DCLK clock domain boundary (CDC) for the DMC fabric.
		0 Sync 1:1
		1 Sync n:1
		2 Sync 1:n
		3 Sync m:n
		4 Async
		5 Reserved
		6 Reserved
		7 Reserved

ADSP-2156x SCB3 Register Descriptions

(SCB3) contains the following registers.

Table 41-114: ADSP-2156x SCB3 Register List

Name	Description
<code>SCB3_APB_DCLK_0_IB_SYNC_MODE</code>	APB DCLK 0 Synchronization Mode Register

APB DCLK 0 Synchronization Mode Register

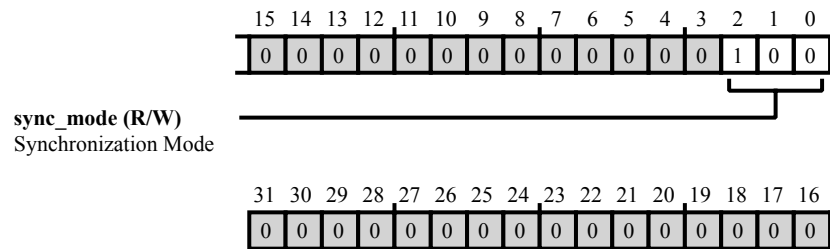


Figure 41-111: SCB3_APB_DCLK_0_IB_SYNC_MODE Register Diagram

Table 41-115: SCB3_APB_DCLK_0_IB_SYNC_MODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	SYNC_MODE	Synchronization Mode. The <code>SCB3_APB_DCLK_0_IB_SYNC_MODE</code> bit field configures the SYSCLK-DCLK clock domain boundary (CDC) for the MMRG fabric.
		0 Sync 1:1
		1 Sync n:1
		2 Sync 1:n
		3 Sync m:n
		4 Async
		5 Reserved
		6 Reserved
		7 Reserved

ADSP-2156x SCB4 Register Descriptions

(SCB4) contains the following registers.

Table 41-116: ADSP-2156x SCB4 Register List

Name	Description
<code>SCB4_COMP_ID_0</code>	Comp Id 0
<code>SCB4_COMP_ID_1</code>	Comp Id 1
<code>SCB4_COMP_ID_2</code>	Comp Id 2
<code>SCB4_COMP_ID_3</code>	Comp Id 3
<code>SCB4_FABRIC_S2PORT_IB_FN_MOD</code>	Fabric S2port Ib.fn Mod

Table 41-116: ADSP-2156x SCB4 Register List (Continued)

Name	Description
SCB4_FABRIC_S2PORT_IB_READ_QOS	Fabric S2port Ib.read Qos
SCB4_FABRIC_S2PORT_IB_WRITE_QOS	Fabric S2port Ib.write Qos
SCB4_FIR_CH0_IB_FN_MOD	Fir Ch0 Ib.fn Mod
SCB4_FIR_CH0_IB_FN_MOD2	Fir Ch0 Ib.fn Mod2
SCB4_FIR_CH0_IB_READ_QOS	Fir Ch0 Ib.read Qos
SCB4_FIR_CH0_IB_WRITE_QOS	Fir Ch0 Ib.write Qos
SCB4_FIR_CH1_IB_FN_MOD	Fir Ch1 Ib.fn Mod
SCB4_FIR_CH1_IB_FN_MOD2	Fir Ch1 Ib.fn Mod2
SCB4_FIR_CH1_IB_READ_QOS	Fir Ch1 Ib.read Qos
SCB4_FIR_CH1_IB_WRITE_QOS	Fir Ch1 Ib.write Qos
SCB4_IIR_CH0_IB_FN_MOD	Iir Ch0 Ib.fn Mod
SCB4_IIR_CH0_IB_FN_MOD2	Iir Ch0 Ib.fn Mod2
SCB4_IIR_CH0_IB_READ_QOS	Iir Ch0 Ib.read Qos
SCB4_IIR_CH0_IB_WRITE_QOS	Iir Ch0 Ib.write Qos
SCB4_IIR_CH1_IB_FN_MOD	Iir Ch1 Ib.fn Mod
SCB4_IIR_CH1_IB_FN_MOD2	Iir Ch1 Ib.fn Mod2
SCB4_IIR_CH1_IB_READ_QOS	Iir Ch1 Ib.read Qos
SCB4_IIR_CH1_IB_WRITE_QOS	Iir Ch1 Ib.write Qos
SCB4_PERIPH_ID_0	Periph Id 0
SCB4_PERIPH_ID_1	Periph Id 1
SCB4_PERIPH_ID_2	Periph Id 2
SCB4_PERIPH_ID_3	Periph Id 3
SCB4_PERIPH_ID_4	Periph Id 4
SCB4_PERIPH_ID_5	Periph Id 5
SCB4_PERIPH_ID_6	Periph Id 6
SCB4_PERIPH_ID_7	Periph Id 7
SCB4_SHARC_DPORT_FN_MOD	Sharc Dport.fn Mod
SCB4_SHARC_DPORT_READ_QOS	Sharc Dport.read Qos
SCB4_SHARC_DPORT_WRITE_QOS	Sharc Dport.write Qos

Comp Id 0

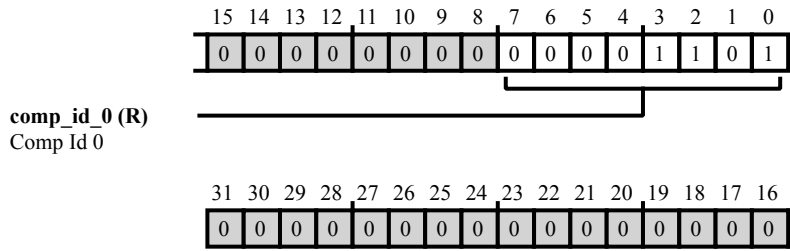


Figure 41-112: SCB4_COMP_ID_0 Register Diagram

Table 41-117: SCB4_COMP_ID_0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_0	Comp Id 0.

Comp Id 1

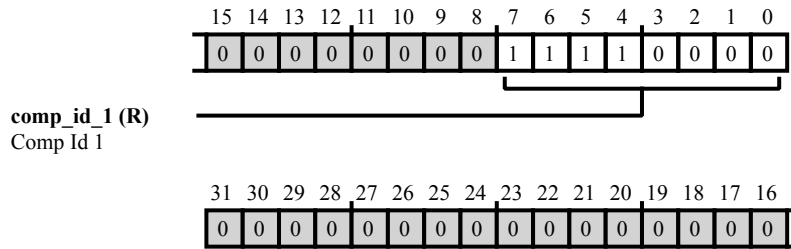


Figure 41-113: SCB4_COMP_ID_1 Register Diagram

Table 41-118: SCB4_COMP_ID_1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_1	Comp Id 1.

Comp Id 2

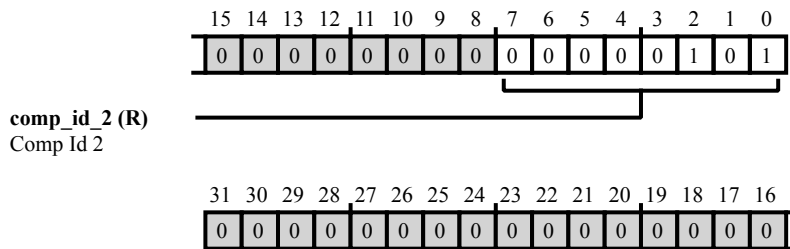


Figure 41-114: SCB4_COMP_ID_2 Register Diagram

Table 41-119: SCB4_COMP_ID_2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_2	Comp Id 2.

Comp Id 3

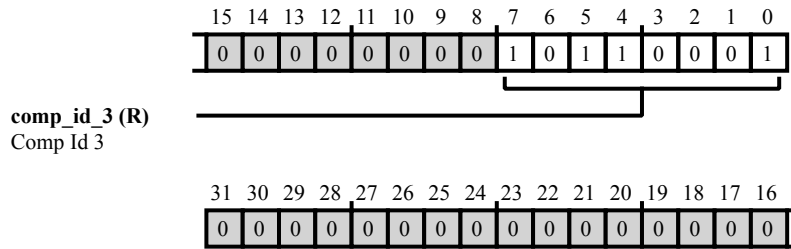


Figure 41-115: SCB4_COMP_ID_3 Register Diagram

Table 41-120: SCB4_COMP_ID_3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_3	Comp Id 3.

Fabric S2port Ib.fn Mod

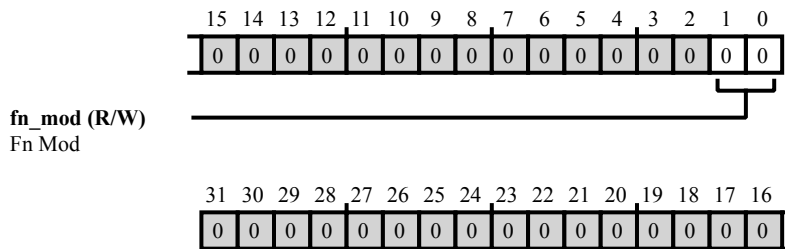


Figure 41-116: SCB4_FABRIC_S2PORT_IB_FN_MOD Register Diagram

Table 41-121: SCB4_FABRIC_S2PORT_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

Fabric S2port Ib.read Qos

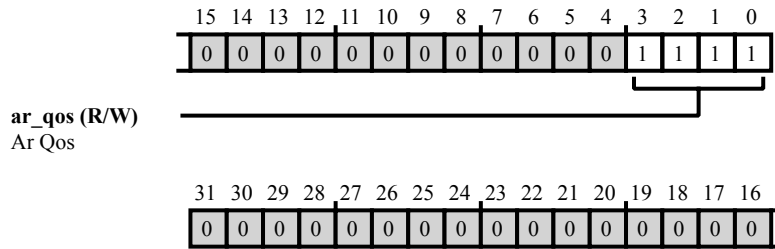


Figure 41-117: SCB4_FABRIC_S2PORT_IB_READ_QOS Register Diagram

Table 41-122: SCB4_FABRIC_S2PORT_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

Fabric S2port Ib.write Qos

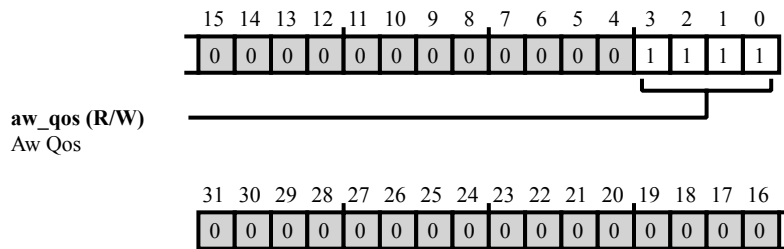


Figure 41-118: SCB4_FABRIC_S2PORT_IB_WRITE_QOS Register Diagram

Table 41-123: SCB4_FABRIC_S2PORT_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

Fir Ch0 Ib.fn Mod

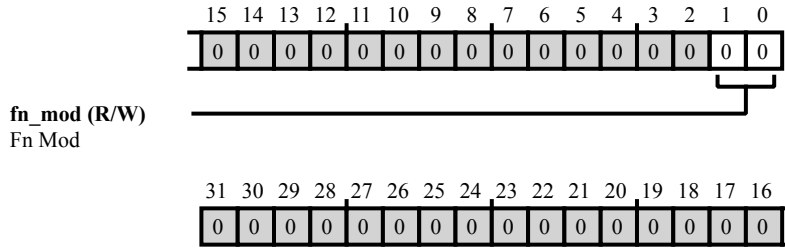


Figure 41-119: SCB4_FIR_CH0_IB_FN_MOD Register Diagram

Table 41-124: SCB4_FIR_CH0_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

Fir Ch0 Ib.fn Mod2

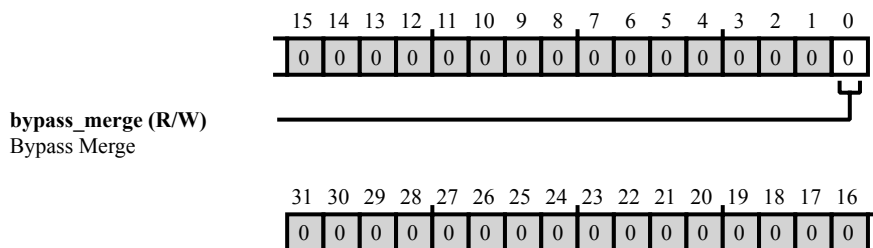


Figure 41-120: SCB4_FIR_CH0_IB_FN_MOD2 Register Diagram

Table 41-125: SCB4_FIR_CH0_IB_FN_MOD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	BYPASS_MERGE	Bypass Merge.

Fir Ch0 Ib.read Qos

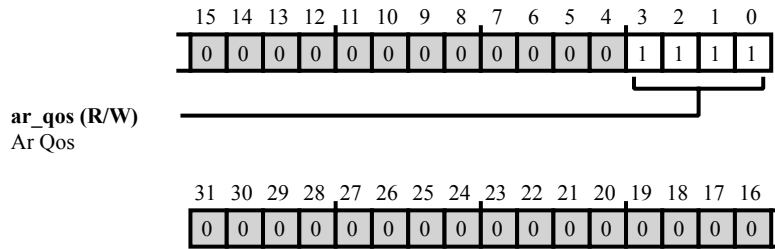


Figure 41-121: SCB4_FIR_CH0_IB_READ_QOS Register Diagram

Table 41-126: SCB4_FIR_CH0_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

Fir Ch0 Ib.write Qos

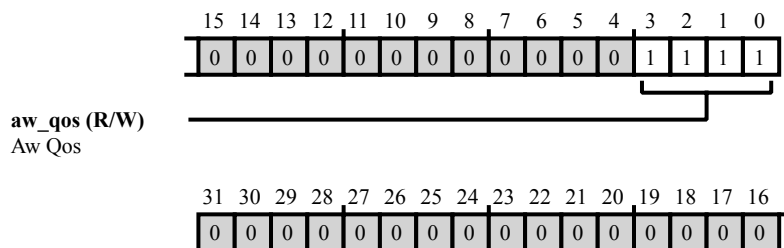


Figure 41-122: SCB4_FIR_CH0_IB_WRITE_QOS Register Diagram

Table 41-127: SCB4_FIR_CH0_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

Fir Ch1 Ib.fn Mod

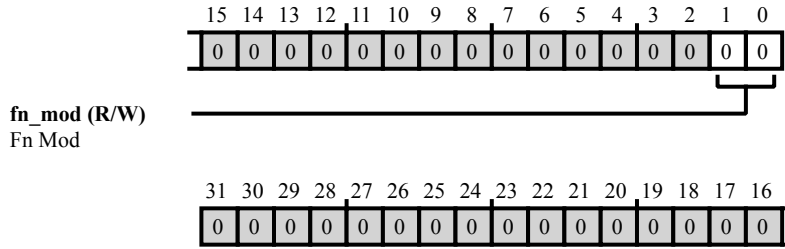


Figure 41-123: SCB4_FIR_CH1_IB_FN_MOD Register Diagram

Table 41-128: SCB4_FIR_CH1_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

Fir Ch1 Ib.fn Mod2

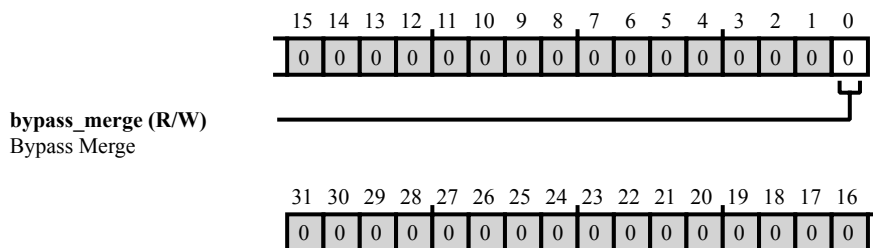


Figure 41-124: SCB4_FIR_CH1_IB_FN_MOD2 Register Diagram

Table 41-129: SCB4_FIR_CH1_IB_FN_MOD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	BYPASS_MERGE	Bypass Merge.

Fir Ch1 Ib.read Qos

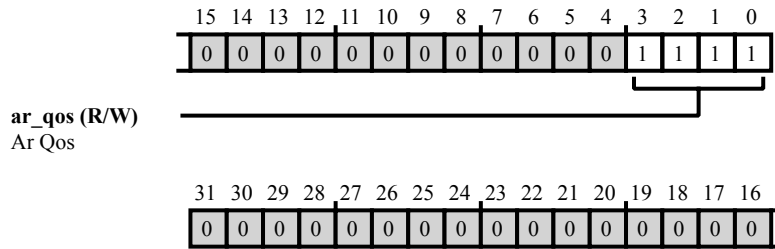


Figure 41-125: SCB4_FIR_CH1_IB_READ_QOS Register Diagram

Table 41-130: SCB4_FIR_CH1_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

Fir Ch1 Ib.write Qos

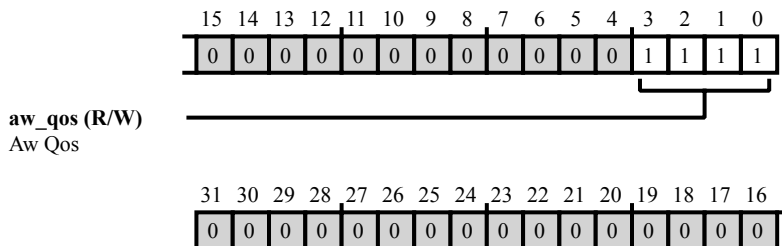


Figure 41-126: SCB4_FIR_CH1_IB_WRITE_QOS Register Diagram

Table 41-131: SCB4_FIR_CH1_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

lir Ch0 Ib.fn Mod

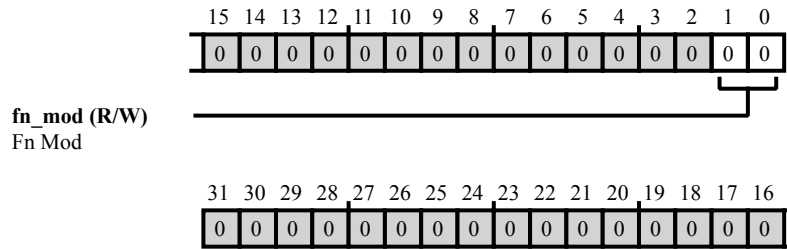


Figure 41-127: SCB4_IIR_CH0_IB_FN_MOD Register Diagram

Table 41-132: SCB4_IIR_CH0_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

lir Ch0 Ib.fn Mod2

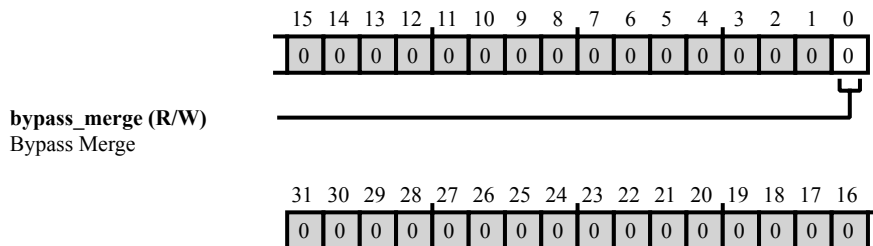


Figure 41-128: SCB4_IIR_CH0_IB_FN_MOD2 Register Diagram

Table 41-133: SCB4_IIR_CH0_IB_FN_MOD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	BYPASS_MERGE	Bypass Merge.

lir Ch0 Ib.read Qos

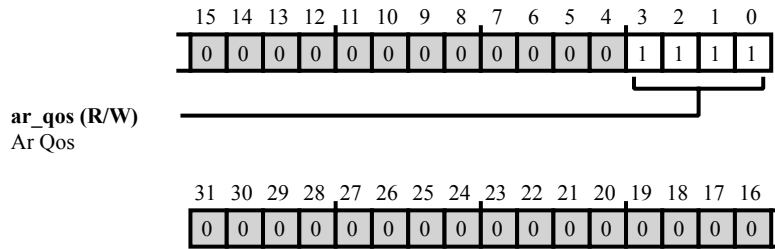


Figure 41-129: SCB4_IIR_CH0_IB_READ_QOS Register Diagram

Table 41-134: SCB4_IIR_CH0_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

lir Ch0 Ib.write Qos

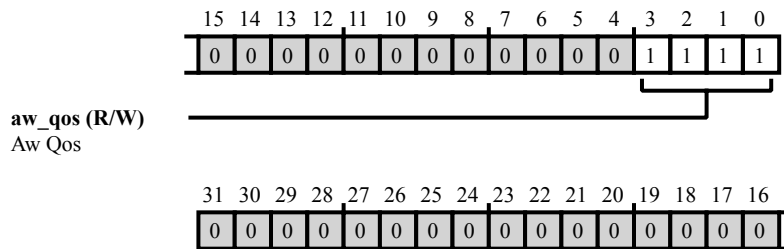


Figure 41-130: SCB4_IIR_CH0_IB_WRITE_QOS Register Diagram

Table 41-135: SCB4_IIR_CH0_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

lir Ch1 Ib.fn Mod

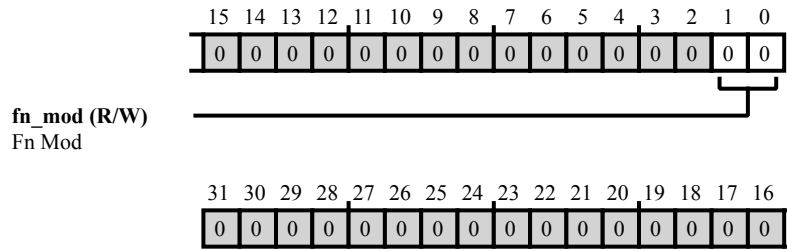


Figure 41-131: SCB4_IIR_CH1_IB_FN_MOD Register Diagram

Table 41-136: SCB4_IIR_CH1_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

lir Ch1 Ib.fn Mod2

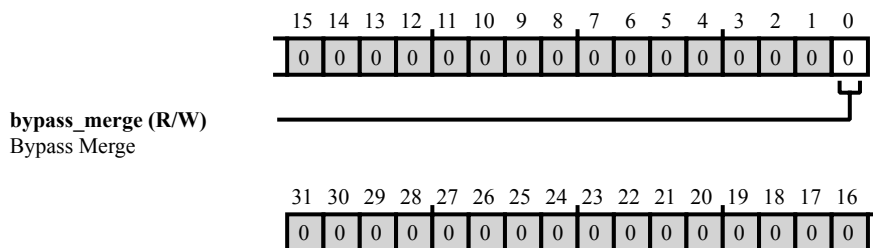


Figure 41-132: SCB4_IIR_CH1_IB_FN_MOD2 Register Diagram

Table 41-137: SCB4_IIR_CH1_IB_FN_MOD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	BYPASS_MERGE	Bypass Merge.

lir Ch1 Ib.read Qos

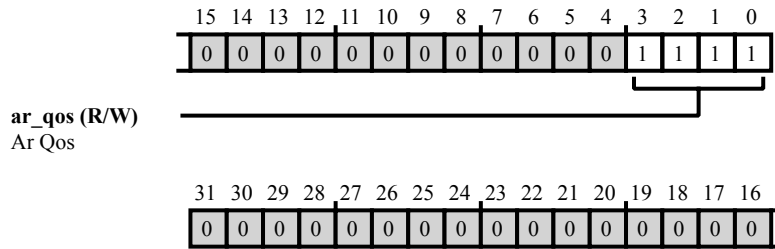


Figure 41-133: SCB4_IIR_CH1_IB_READ_QOS Register Diagram

Table 41-138: SCB4_IIR_CH1_IB_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

lir Ch1 Ib.write Qos

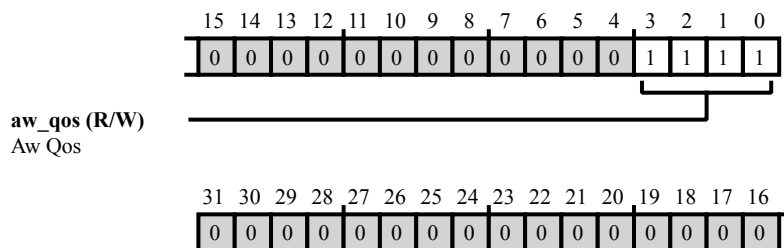


Figure 41-134: SCB4_IIR_CH1_IB_WRITE_QOS Register Diagram

Table 41-139: SCB4_IIR_CH1_IB_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

Periph Id 0

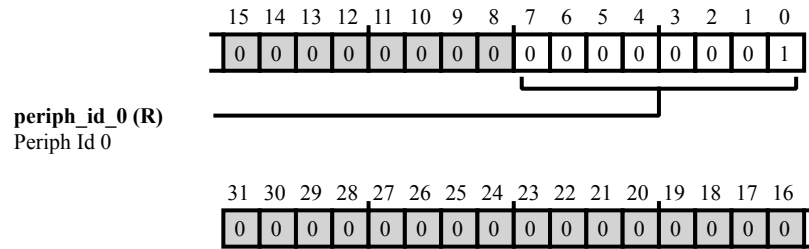


Figure 41-135: SCB4_PERIPH_ID_0 Register Diagram

Table 41-140: SCB4_PERIPH_ID_0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_0	Periph Id 0.

Periph Id 1

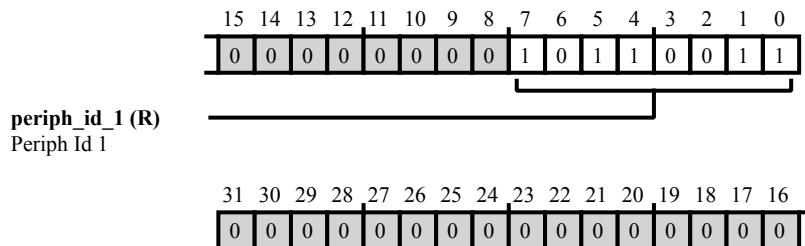


Figure 41-136: SCB4_PERIPH_ID_1 Register Diagram

Table 41-141: SCB4_PERIPH_ID_1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_1	Periph Id 1.

Periph Id 2

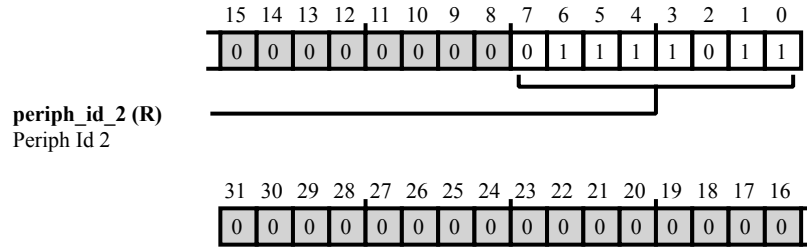


Figure 41-137: SCB4_PERIPH_ID_2 Register Diagram

Table 41-142: SCB4_PERIPH_ID_2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_2	Periph Id 2.

Periph Id 3

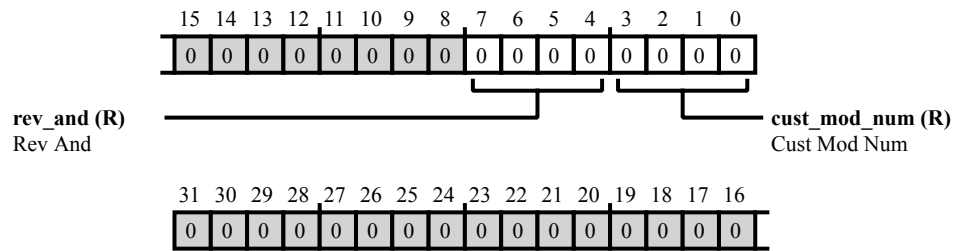


Figure 41-138: SCB4_PERIPH_ID_3 Register Diagram

Table 41-143: SCB4_PERIPH_ID_3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV_AND	Rev And.
3:0 (R/NW)	CUST_MOD_NUM	Cust Mod Num.

Periph Id 4

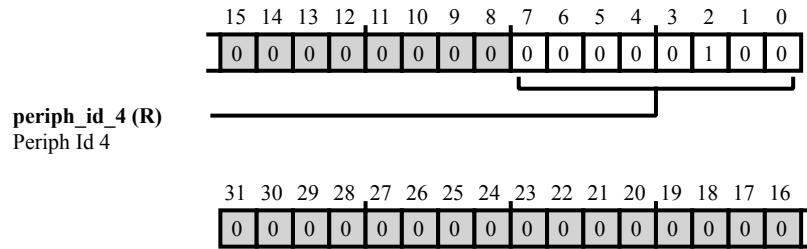


Figure 41-139: SCB4_PERIPH_ID_4 Register Diagram

Table 41-144: SCB4_PERIPH_ID_4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_4	Periph Id 4.

Periph Id 5

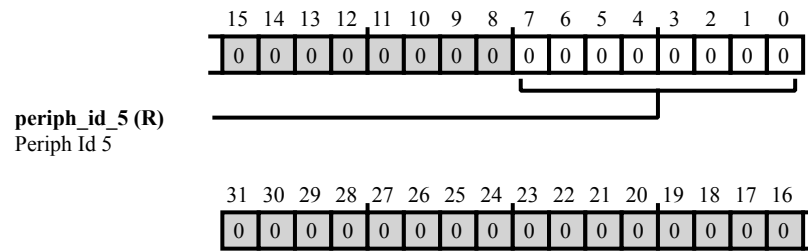


Figure 41-140: SCB4_PERIPH_ID_5 Register Diagram

Table 41-145: SCB4_PERIPH_ID_5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_5	Periph Id 5.

Periph Id 6

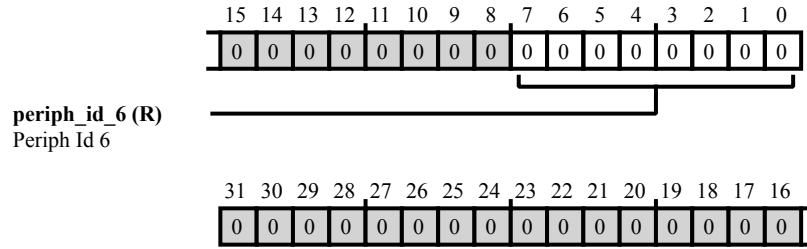


Figure 41-141: SCB4_PERIPH_ID_6 Register Diagram

Table 41-146: SCB4_PERIPH_ID_6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_6	Periph Id 6.

Periph Id 7

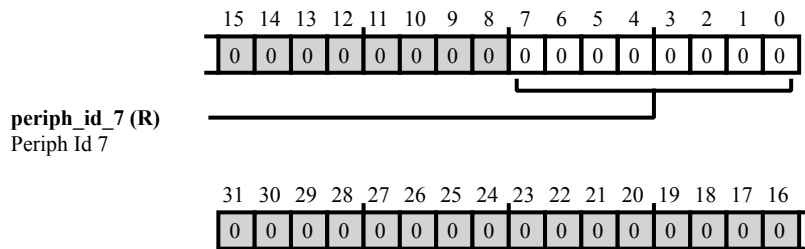


Figure 41-142: SCB4_PERIPH_ID_7 Register Diagram

Table 41-147: SCB4_PERIPH_ID_7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_7	Periph Id 7.

Sharc Dport.fn Mod

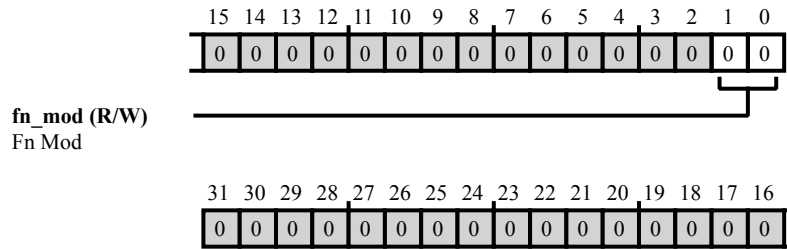


Figure 41-143: SCB4_SHARC_DPORT_FN_MOD Register Diagram

Table 41-148: SCB4_SHARC_DPORT_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

Sharc Dport.read Qos

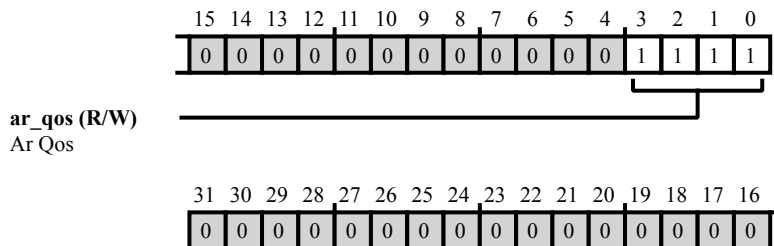


Figure 41-144: SCB4_SHARC_DPORT_READ_QOS Register Diagram

Table 41-149: SCB4_SHARC_DPORT_READ_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AR_QOS	Ar Qos.

Sharc Dport.write Qos

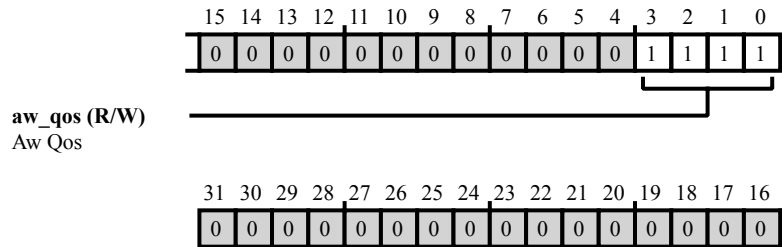


Figure 41-145: SCB4_SHARC_DPORT_WRITE_QOS Register Diagram

Table 41-150: SCB4_SHARC_DPORT_WRITE_QOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	AW_QOS	Aw Qos.

ADSP-2156x SCB5 Register Descriptions

(SCB5) contains the following registers.

Table 41-151: ADSP-2156x SCB5 Register List

Name	Description
SCB5_COMP_ID_0	Comp Id 0
SCB5_COMP_ID_1	Comp Id 1
SCB5_COMP_ID_2	Comp Id 2
SCB5_COMP_ID_3	Comp Id 3
SCB5_PERIPH_ID_0	Periph Id 0
SCB5_PERIPH_ID_1	Periph Id 1
SCB5_PERIPH_ID_2	Periph Id 2
SCB5_PERIPH_ID_3	Periph Id 3
SCB5_PERIPH_ID_4	Periph Id 4
SCB5_PERIPH_ID_5	Periph Id 5
SCB5_PERIPH_ID_6	Periph Id 6
SCB5_PERIPH_ID_7	Periph Id 7
SCB5_REMAP	
SCB5_SPI2_S_IB_FN_MOD	Spi2 S Ib.fn Mod

Table 41-151: ADSP-2156x SCB5 Register List (Continued)

Name	Description
SCB5_SPI2_S_IB_FN_MOD_ISS_BM	Spi2 S Ib.fn Mod Iss Bm
SCB5_SPIF_M_FN_MOD	Spif M.fn Mod

Comp Id 0

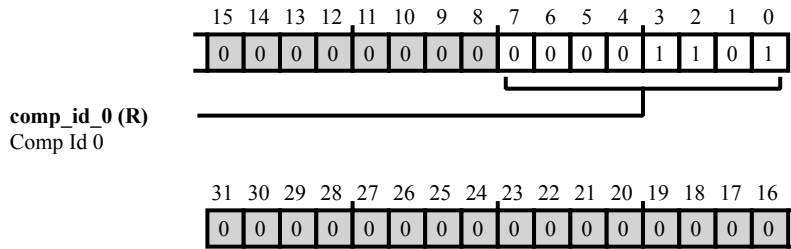


Figure 41-146: SCB5_COMP_ID_0 Register Diagram

Table 41-152: SCB5_COMP_ID_0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_0	Comp Id 0.

Comp Id 1

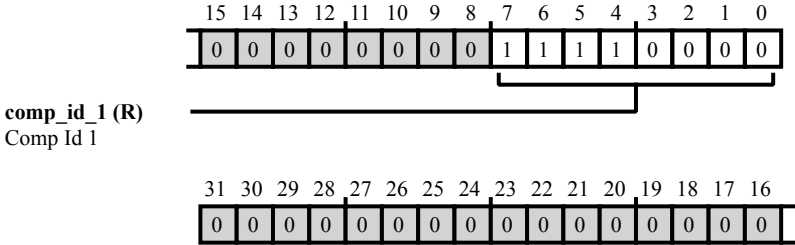


Figure 41-147: SCB5_COMP_ID_1 Register Diagram

Table 41-153: SCB5_COMP_ID_1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_1	Comp Id 1.

Comp Id 2

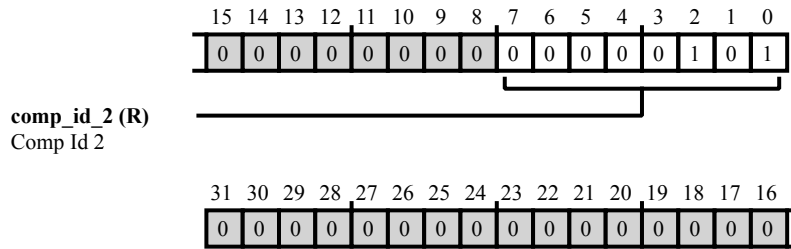


Figure 41-148: SCB5_COMP_ID_2 Register Diagram

Table 41-154: SCB5_COMP_ID_2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_2	Comp Id 2.

Comp Id 3

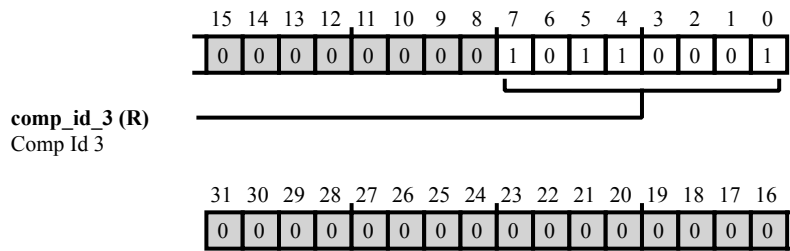


Figure 41-149: SCB5_COMP_ID_3 Register Diagram

Table 41-155: SCB5_COMP_ID_3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMP_ID_3	Comp Id 3.

Periph Id 0

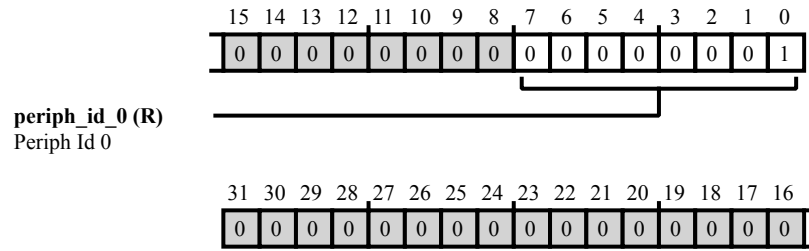


Figure 41-150: SCB5_PERIPH_ID_0 Register Diagram

Table 41-156: SCB5_PERIPH_ID_0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_0	Periph Id 0.

Periph Id 1

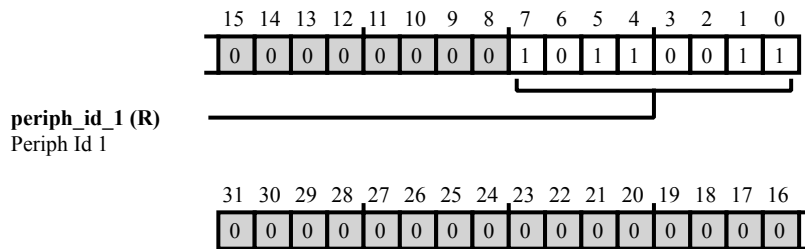


Figure 41-151: SCB5_PERIPH_ID_1 Register Diagram

Table 41-157: SCB5_PERIPH_ID_1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_1	Periph Id 1.

Periph Id 2

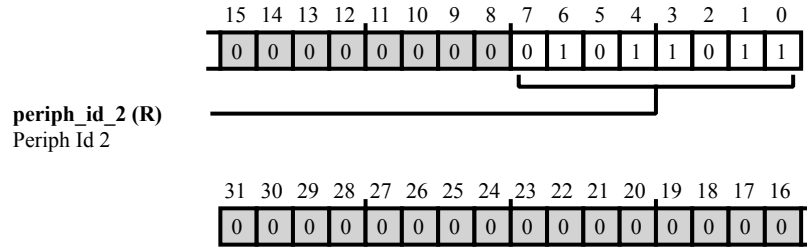


Figure 41-152: SCB5_PERIPH_ID_2 Register Diagram

Table 41-158: SCB5_PERIPH_ID_2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_2	Periph Id 2.

Periph Id 3

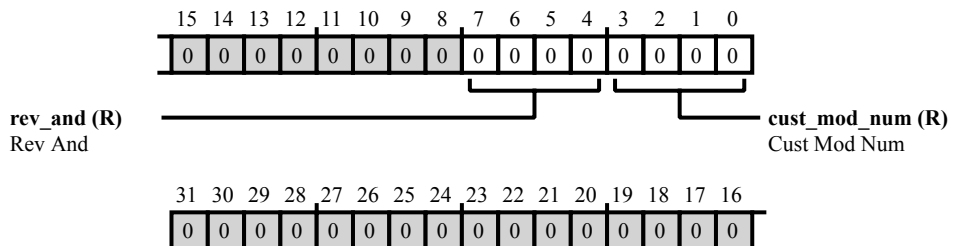


Figure 41-153: SCB5_PERIPH_ID_3 Register Diagram

Table 41-159: SCB5_PERIPH_ID_3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV_AND	Rev And.
3:0 (R/NW)	CUST_MOD_NUM	Cust Mod Num.

Periph Id 4

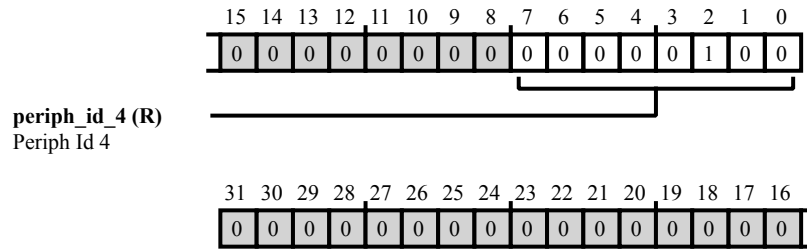


Figure 41-154: SCB5_PERIPH_ID_4 Register Diagram

Table 41-160: SCB5_PERIPH_ID_4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_4	Periph Id 4.

Periph Id 5

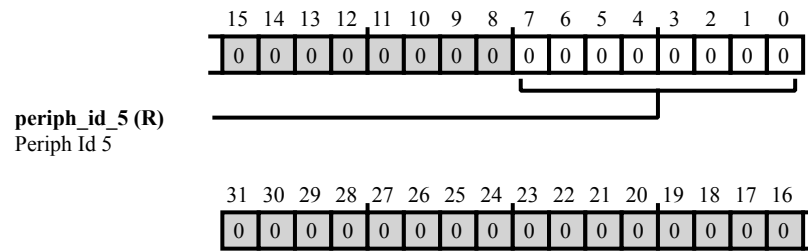


Figure 41-155: SCB5_PERIPH_ID_5 Register Diagram

Table 41-161: SCB5_PERIPH_ID_5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_5	Periph Id 5.

Periph Id 6

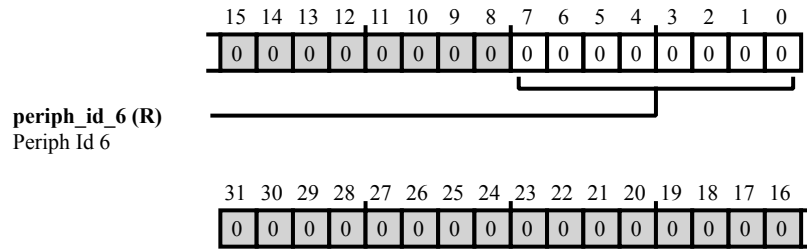


Figure 41-156: SCB5_PERIPH_ID_6 Register Diagram

Table 41-162: SCB5_PERIPH_ID_6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_6	Periph Id 6.

Periph Id 7

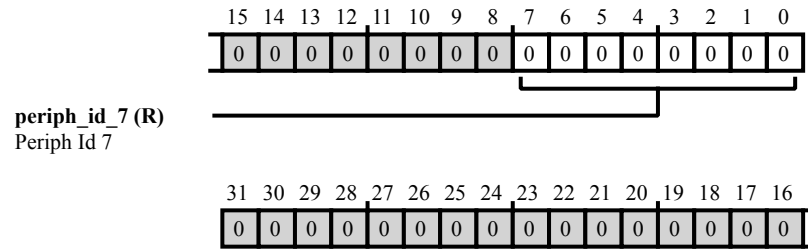


Figure 41-157: SCB5_PERIPH_ID_7 Register Diagram

Table 41-163: SCB5_PERIPH_ID_7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PERIPH_ID_7	Periph Id 7.

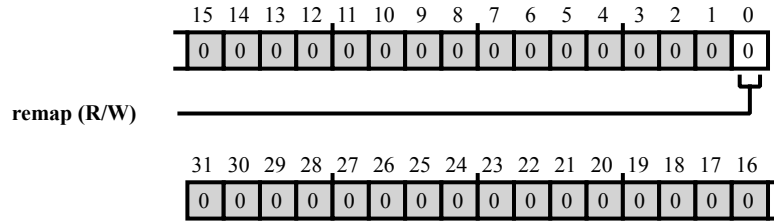


Figure 41-158: SCB5_REMAP Register Diagram

Table 41-164: SCB5_REMAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	REMAP	.

Spi2 S Ib.fn Mod

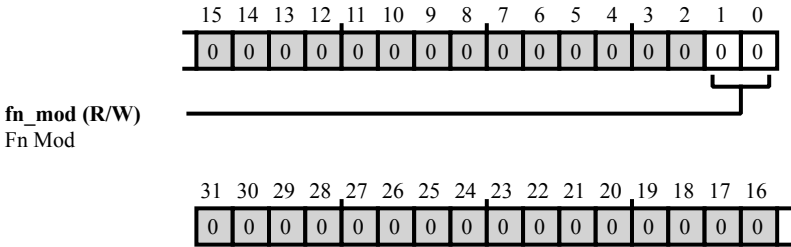


Figure 41-159: SCB5_SPI2_S_IB_FN_MOD Register Diagram

Table 41-165: SCB5_SPI2_S_IB_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

Spi2 S Ib.fn Mod Iss Bm

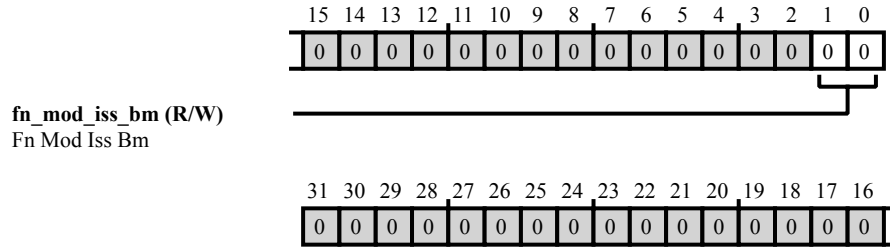


Figure 41-160: SCB5_SPI2_S_IB_FN_MOD_ISS_BM Register Diagram

Table 41-166: SCB5_SPI2_S_IB_FN_MOD_ISS_BM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD_ISS_BM	Fn Mod Iss Bm.

Spif M.fn Mod

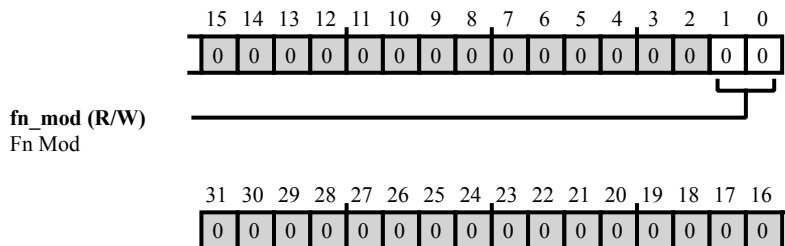


Figure 41-161: SCB5_SPIF_M_FN_MOD Register Diagram

Table 41-167: SCB5_SPIF_M_FN_MOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	FN_MOD	Fn Mod.

42 System Watchpoint Unit (SWU)

The system watchpoint unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

Each SWU contains four match groups of registers with associated hardware. These four SWU match groups operate independently, but share common event (interrupt and trigger) outputs. Each match group can monitor either the write or read address channel and can operate in either watchpoint mode or bandwidth mode.

SWU Features

The system watchpoint unit has the following features.

- Four independent match groups for each SWU
- Each match group can operate in either bandwidth mode or watchpoint mode

SWU Functional Description

This section describes the function of the SWU match block, interface block, and MMR block.

ADSP-2156x SWU Register List

The System Watchpoint Unit (SWU) provides debug and development support through flexible transaction level and bandwidth monitoring and associated event triggering. The SWU can generate events based on monitoring transactions at the system slaves through watchpoint-match groups. The SWU also provides watchpoint event status reporting, a global lock, and processor reset capability. A set of registers governs SWU operations. For more information on SWU functionality, see the SWU register descriptions.

Table 42-1: ADSP-2156x SWU Register List

Name	Description
SWU_CNT [n]	Count Register n
SWU_CTL [n]	Control Register n

Table 42-1: ADSP-2156x SWU Register List (Continued)

Name	Description
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

42.2.2 SWU Interrupt List

42.2.3 SWU Trigger List

SWU Definitions

The following definitions are helpful when using the SWU module.

Watchpoint Mode

Mode in which transactions are recognized on an exact match. Actions can be configured to be taken after a specified number of matches have occurred.

Bandwidth Mode

Mode in which transactions are recognized and counted inside sampling window.

SWU Architectural Concepts

The information in this section provides basic module design concepts.

SWU-to-SCB Interface

The SWU system crossbar interface block latches all transactions on the system crossbar read and write address channels when the SWU_GCTL.EN register enable bit is set.

SWU Block Diagram

The *System Watchpoint Unit Top-Level Block Diagram* figure shows the SWU block diagram.

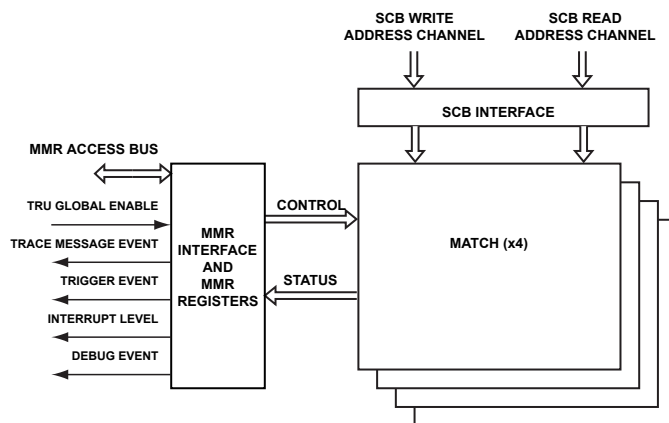


Figure 42-1: System Watchpoint Unit Top-Level Block Diagram

SCB Interface Block

The SWU system crossbar (SCB) latches all transactions on the SCB read and write address channels when the `SWU_GCTL.EN` bit is set.

MMR Interface Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupt requests and events from each match block into common outputs.

SWU Operating Modes

There are two operating modes supported by the SWU: bandwidth mode and watchpoint mode.

Bandwidth Mode

In bandwidth mode, the SWU module counts transactions which match the properties specified in the `SWU_CTL[n]` register during a sampling window determined by the respective `SWU_CNT[n]` register. At the end of the sampling window, the SWU stores results in the `SWU_HIST[n]` register. If the sampled bandwidth falls outside a programmed range, then the programmed action occurs.

Watchpoint Mode

In watchpoint mode, if the `SWU_CTL[n].CNTEN` bit is set, the SWU module decrements the `SWU_CUR[n]` register for each match, until it equals zero, at which point any programmed actions occur. The `SWU_CUR[n]` register is then reloaded from the `SWU_CNT[n]` register (if the `SWU_CTL[n].CNTRPTEN` bit is set), and the cycle repeats. If the `SWU_CTL[n].CNTRPTEN` bit is not set, any programmed actions happen on every match.

Match Block

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive or exclusive range), ID (with masking), and security. All enabled matches are AND'ed together to determine a match.

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive or exclusive range), ID (with masking), security, and lock type. All enabled matches are AND'ed together to determine a match.

Scaling

Scaling allows the SWU to count more transactions by scaling the number of transactions and the number of clock cycles in bandwidth window (CNTn register) by 10,100 or 1000. This functionality is applicable only in bandwidth mode (SWU_CTL[n].BWEN==1).

Consider a case where the SWU_TARG[n].BWMAX bit field is programmed to 2. In the absence of any scaling, bandwidth overflow occurs when the SWU_CUR[n].CURBW bit field value > the SWU_TARG[n].BWMAX bit field value (2). With scaling set to 1:100 and after 275 transactions, the SWU_CUR[n].CURBW value is still 2 and equal to SWU_TARG[n].BWMAX. This event triggers a bandwidth overflow as the actual number of transactions is greater than 200 (2×100). The code can be rerun with a smaller scaling selected to analyze the cause for the overflow.

The counter increments after every group of N transactions with scaling enabled, where N is either 10,100 or 1000. (For N = 10, 0–9 transactions == 0 scaled transaction, 10–19 transactions == 1 scaled transaction, 20–29 transactions == 2 scaled transactions, and so on).

Fractional counts with scaling enabled are discarded and not rolled over from one bandwidth window to the next. For example, consider a case where scaling by 1000 is configured and the first window has 1200 transactions. The second window has 2800 transactions. The bandwidth for the first window is read as 1 and the second as 2. The 200 transactions from the first window do not get carried forward to the next window.

Do not use scaling by 10 with the SWU_CTL[n].BLENINC bit enabled if any of the masters accessing the slave can launch a transaction of burst length 16.

SWU Event Control

The SWU can generate the following events when a match occurs and when the event is enabled by configuring the proper bits in the control register.

1. Trace Message
2. Trigger
3. Interrupt request
4. Debug

SWU Interrupts

All interrupt requests and events from each match block are merged into common outputs.

SWU Status and Errors

SWU status and errors are reported in the `SWU_GSTAT` register. The SWU records an address error when a write or read attempt is made to the MMR address space of the SWU and the register does not exist. This error is the only one the SWU records. The register contains bits that perform the following functions.

- Indicate whether a particular match group sampled a transaction that is below a minimum target or above a maximum target in bandwidth mode.
- Indicate whether a watchpoint match occurred for each match group.
- Indicate whether an interrupt request was triggered due to a match event from one of the match groups.

Triggers

The SWU can be either a trigger master or a trigger slave depending on the trigger routing unit (TRU) configuration. As a trigger master, programs must set the `SWU_CTL[n].TRGEN` bit so that when a match condition is met, a trigger event is generated. Each SWU in the system can also be a trigger slave when mapped as one in the TRU.

When the SWU is a slave, a trigger event activates the SWU by automatically setting the `SWU_GCTL.EN` bit. Since the SWU can be automatically enabled through a trigger event, programs must pre-configure the SWU before enabling the TRU. Furthermore, although a trigger event can enable the SWU as a slave, to disable the SWU, programs must manually clear the `SWU_GCTL.EN` bit.

SWU Programming Model

Program the appropriate registers to use the SWU. Each control register configures aspects such as:

- The direction of monitoring (reads or writes)
- Whether SWU uses bandwidth mode or watchpoint mode
- The setup of events that are triggered when a condition is met while monitoring using the SWU

Configure supplemental registers such as the lower (`SWU_LA[n]`) and upper (`SWU_UA[n]`) address boundaries before enabling the SWU.

Once the SWU has been enabled and the monitoring conditions are met, events are generated when configured.

The global status register (`SWU_GSTAT`) can be read to observe the status of the units.

The *SWU Logical Flow* diagram shows the logical program flow of the SWU.

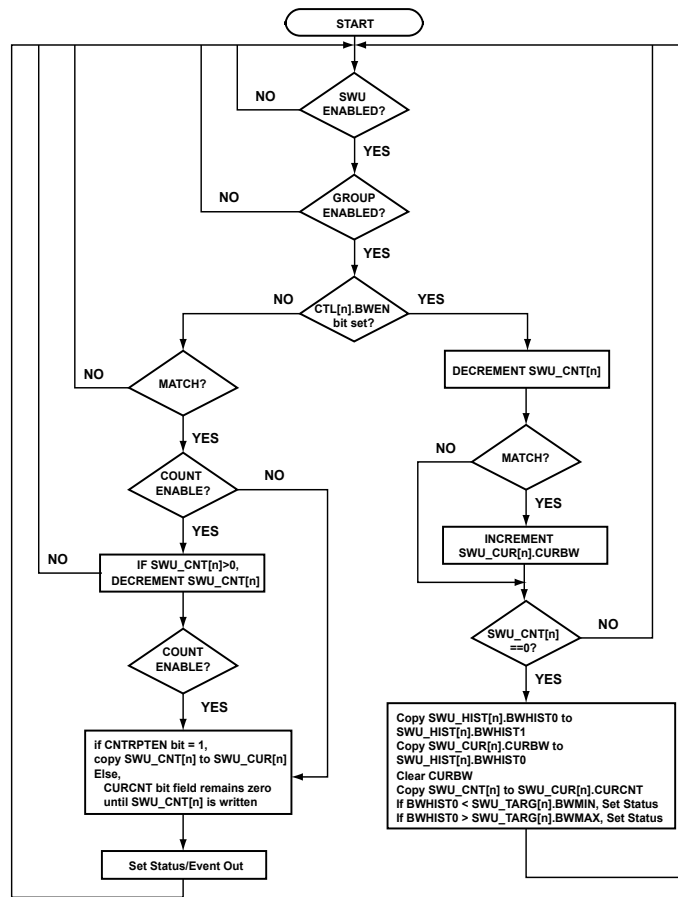


Figure 42-2: SWU Logical Flow

SWU Mode Configuration

The following sections show the steps for configuring SWU bandwidth mode and watchpoint mode.

Configuring the SWU for Bandwidth Mode

In bandwidth mode, the SWU counts transactions which match during a sampling window. At the end of the sampling window, the SWU stores the results. An action can be taken if the sampled bandwidth goes above or falls below a programmed range.

1. Configure the `SWU_CTL[n].DIR` bit to test the match on writes or reads.
2. Configure the `SWU_CTL[n].ACMPM` bits to address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the `SWU_CTL[n].IDCMPEN` bit.
4. Set the `SWU_CTL[n].BLENINC` bit to increment by burst length or clear it to increment by 1.
5. Configure the `SWU_CTL[n].MAXACT` and `SWU_CTL[n].MINACT` bits to enable actions taken when the bandwidth goes above the maximum, or falls below the minimum, respectively.

6. Set the `SWU_CTL[n].BWEN=1` to enable bandwidth mode.
7. Program the lower address register, `SWU_LA[n]`, and upper address register, `SWU_UA[n]`, to define the memory range for comparison.
8. If ID comparison is enabled, program the ID register, `SWU_ID[n]`.
9. Program the count register, `SWU_CNT[n]`, with the number of clock cycles for which the SWU counts the number of matches.
10. If the SWU is set to respond when the bandwidth measurement underflows or overflows, program the min and max values into the `SWU_TARG[n]` register.
11. Enable the SWU

The SWU counts the number of matches in a pre-defined number of clock cycles as programmed. As an option, it can define lower and upper limits. If the matches fall outside the limits, an action can be taken.

Configuring the SWU for Watchpoint Mode

In watchpoint mode, the SWU can trigger a programmed action after every match or after a number of matches. This sequence can be automatically reset.

1. Set the `SWU_CTL[n].DIR` bit to test the match on writes or reads.
2. Configure the `SWU_CTL[n].ACMPM` bits for address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the `SWU_CTL[n].IDCMPEN`.
4. Set the `SWU_CTL[n].CNTEN` bit to enable the events to be triggered when the count decrements to zero.
5. If needed, set the `SWU_CTL[n].CNTRPTEN` bit to automatically reload the counter after it has decremented to zero to start another match sequence.
6. Clear the `SWU_CTL[n].BWEN = 0` to configure watchpoint mode.
7. Configure the lower address register, `SWU_LA[n]`, and upper address register, `SWU_UA[n]`, to define the memory range for comparison.
8. If ID comparison is enabled, configure the ID register, `SWU_ID[n]`.
9. Configure the count register, `SWU_CNT[n]`, to determine how many matches occur before the watchpoint group responds.
10. Enable the SWU.

The SWU detects and counts down the number of match occurrences. When the counter expires, an action is taken.

ADSP-2156x SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 42-2: ADSP-2156x SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

Count Register n

The SWU count registers ($SWU_CNT[n]$) contain a 16-bit count field ($SWU_CNT[n].COUNT$) whose usage differs depending on the mode of the watchpoint group. In bandwidth mode, the $SWU_CNT[n].COUNT$ field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the $SWU_CNT[n].COUNT$ field value determines how many matches occur before the watchpoint group takes action.

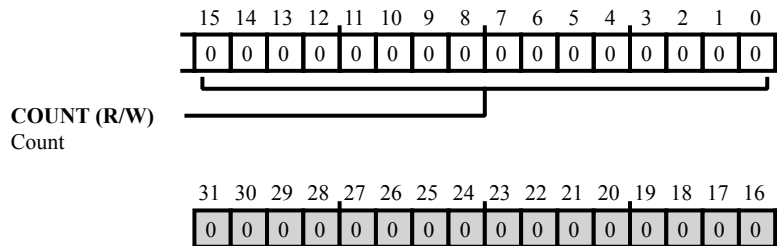


Figure 42-3: $SWU_CNT[n]$ Register Diagram

Table 42-3: $SWU_CNT[n]$ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count. The $SWU_CNT[n].COUNT$ field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the $SWU_CNT[n].COUNT$ field value determines how many matches occur before the watchpoint group takes action.

Control Register n

The SWU control registers (`SWU_CTL[n]`) contain watchpoint attribute controls for all four watchpoint groups. These controls include enabling watchpoints, selecting the transaction direction for match, selecting address comparison mode, enabling ID comparison, enabling security comparison, enabling cycle count, enabling count repeat, enabling debug events, enabling interrupts, enabling triggers, enabling trace messages, enabling bandwidth mode, selecting the burst length increment, and enabling bandwidth underflow and overflow detection.

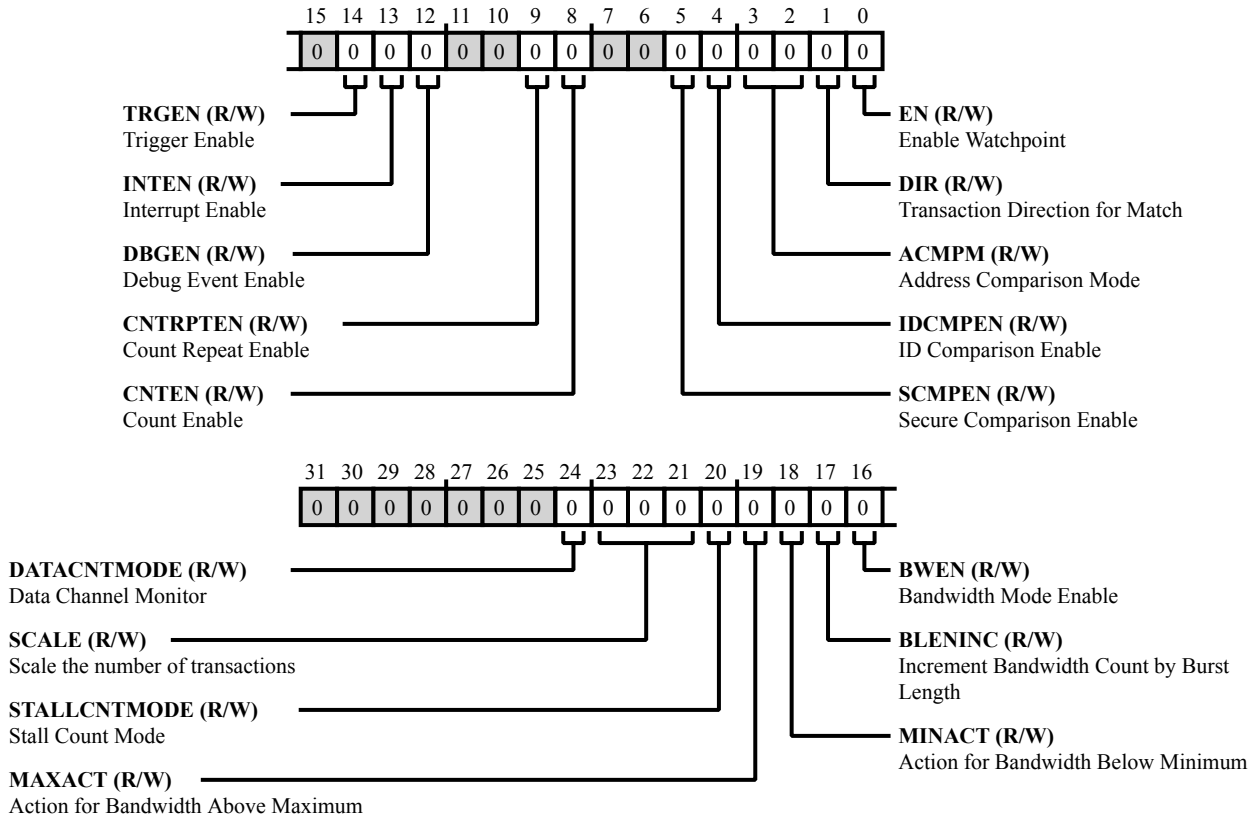


Figure 42-4: SWU_CTL[n] Register Diagram

Table 42-4: SWU_CTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	DATAcntMODE	Data Channel Monitor. The SWU_CTL[n].DATAcntMODE bit determines whether an address channel or a data channel is monitored. Note that in data channel only ID,READY and VALID signals are monitored and hence other comparisons (Address, Lock, Secure) will be ignored even if enabled. SWU_CTL[n].STALLcntMODE and SWU_CTL[n].DIR can be used in conjunction with this bit
		0 Monitor address channel
		1 Monitor data channel
23:21 (R/W)	SCALE	Scale the number of transactions. The SWU_CTL[n].SCALE bit field allows a program to count more transactions by scaling the number of transactions and also the number of clock cycles in the bandwidth window (SWU_CNT[n] register) by 10,100 or 1000. This is applicable only in bandwidth mode (SWU_CTL[n].BWEN==1).
		0 No scaling
		1 1:10
		2 1:100
		3 Reserved
		4 1:1000
		5-7 Reserved
20 (R/W)	STALLcntMODE	Stall Count Mode. The SWU_CTL[n].STALLcntMODE bit determines whether the number of stalls are counted or whether the number of transactions are counted. This feature is only valid in bandwidth mode.
		0 Count number of transactions
		1 Count number of stalls
19 (R/W)	MAXACT	Action for Bandwidth Above Maximum. Each SWU_CTL[n].MAXACT bit determines whether a watchpoint group takes action on bandwidth overflow. This feature is only valid in bandwidth mode.
		0 No Action
		1 Take Action
18 (R/W)	MINACT	Action for Bandwidth Below Minimum. Each SWU_CTL[n].MINACT bit determines whether a watchpoint group takes action on bandwidth underflow. This feature is only valid in bandwidth mode.
		0 No Action
		1 Take Action

Table 42-4: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	BLENINC	<p>Increment Bandwidth Count by Burst Length.</p> <p>Each SWU_CTL[n].BLENINC bit controls how a watchpoint group's bandwidth count is incremented in the SWU_CUR[n] register's SWU_CUR[n].CURBW field. If the SWU_CTL[n].BLENINC bit is cleared (= 0), the SWU increments the bandwidth count by 1 for each matching transaction. If the SWU_CTL[n].BLENINC bit is set (=1), the SWU increments the bandwidth count by the burst length of the transaction for each matching transaction. This feature is only valid for bandwidth mode (SWU_CTL[n].BWEN bit == 1).</p> <p>Note that if the address range match is enabled (SWU_CTL[n].ACMPM bits) and if any address of a burst falls within the address range, the SWU_CUR[n].CURBW field is incremented by the burst length even if some of the burst address fall outside of the range.</p> <p>Also, note that the burst size of the transaction is not included in the increment, only the burst length of the transaction. This increment operation provides an approximate (not exact) number of bus cycles consumed during the bandwidth.</p>
		0 Increment by 1
		1 Burst Length Increment for Bandwidth Count
16 (R/W)	BWEN	<p>Bandwidth Mode Enable.</p> <p>Each SWU_CTL[n].BWEN bit controls whether a watchpoint group operates in watchpoint mode or bandwidth mode. In watchpoint mode, the SWU_CTL[n].CNTEN and (optionally) SWU_CTL[n].CNTRPTEN registers control usage of the cycle count for watchpoint group operations. In bandwidth mode, the SWU_CTL[n].BLENINC, SWU_TARG[n], and SWU_HIST[n] registers control usage of watchpoint matches for watchpoint group operations.</p>
		0 Watchpoint Mode
		1 Bandwidth Mode
14 (R/W)	TRGEN	<p>Trigger Enable.</p> <p>Each SWU_CTL[n].TRGEN bit controls whether a match for a watchpoint group generates a trigger event. This feature is valid in both bandwidth and watchpoint modes.</p>
		0 Disable
		1 Enable
13 (R/W)	INTEN	<p>Interrupt Enable.</p> <p>Each SWU_CTL[n].INTEN bit controls whether a match for a watchpoint group generates an interrupt. This feature is valid in both bandwidth and watchpoint modes.</p>
		0 Disable
		1 Enable

Table 42-4: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	DBGEN	Debug Event Enable. Each SWU_CTL[n].DBGEN bit controls debug event comparison for a watchpoint group, permitting matches based on debug status.
		0 Disable
		1 Enable
9 (R/W)	CNTRPTEN	Count Repeat Enable. Each SWU_CTL[n].CNTRPTEN bit controls whether the watchpoint group's cycle count is reloaded and repeated after cycle countdown. If the SWU_CTL[n] register's SWU_CTL[n].CNTRPTEN bit is set, the SWU_CUR[n] register's SWU_CUR[n].CURCNT field is reloaded from SWU_CNT[n] register's SWU_CNT[n].COUNT field, and the countdown starts again. If SWU_CTL[n].CNTRPTEN bit is cleared, the expired count remains zero, and no further events are signalled. (See the SWU_CTL[n].CNTEN bit description for information regarding the countdown setup.)
		0 Disable
		1 Enable
8 (R/W)	CNTEN	Count Enable. Each SWU_CTL[n].CNTEN bit controls whether the cycle count in the watchpoint group's SWU_CNT[n] register is decremented each cycle until it reaches zero. This feature is only valid in watchpoint mode (SWU_CTL[n].BWEN bit == 0). When the count reaches zero, any enabled watchpoint events are triggered. (See the SWU_CTL[n].CNTRPTEN bit description for optional actions at that may occur at the end of the countdown.)
		0 Disable
		1 Enable
5 (R/W)	SCMPEN	Secure Comparison Enable. Each SWU_CTL[n].SCMPEN bit controls secure transaction comparison operation of an SWU watchpoint group, permitting matches based on transaction security.
		0 Match on all transaction
		1 Match only secure transactions
4 (R/W)	IDCMPEN	ID Comparison Enable. Each SWU_CTL[n].IDCMPEN bit controls the ID comparison operation of an SWU watchpoint group. The ID match is based on comparison with the value in the SWU_ID[n] register.

Table 42-4: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:2 (R/W)	ACMPM	Address Comparison Mode. Each set of SWU_CTL[n].ACMPM bits control the address comparison operation of an SWU watchpoint group. The address within range for comparison is defined as (SWU_LA[n] register <= address < SWU_UA[n] register). The address outside range for comparison is defined as (address < SWU_LA[n]) or (SWU_UA[n] <= address).
		0 No address comparison
		1 Exact match on LAN
		2 Match on address within range
		3 Match on address outside range
1 (R/W)	DIR	Transaction Direction for Match. Each SWU_CTL[n].DIR bit determines whether the SWU check reads or writes for watchpoint matches.
		0 Match on reads only
		1 Match on writes only
0 (R/W)	EN	Enable Watchpoint. Each SWU_CTL[n].EN bit controls the operation of one SWU watchpoint group. Clearing the SWU_CTL[n].EN bit halts the execution of watchpoint or bandwidth tracking operations in the watchpoint group without resetting status or configuration registers. Setting the SWU_CTL[n].EN bit enables the SWU watchpoint group to begin or resume operation with the current configuration and status.
		0 Disable
		1 Enable

Current Register n

The SWU current register ($SWU_CUR[n]$) operation varies depending whether the watchpoint group is in bandwidth mode or watchpoint mode. In both modes, the watchpoint count begins when the SWU loads the register's $SWU_CUR[n].CURCNT$ field from the $SWU_CNT[n]$ register's $SWU_CNT[n].COUNT$ field when the watchpoint count is enabled ($SWU_CTL[n]$ register, $SWU_CTL[n].CNTEN$ bit =1).

In bandwidth mode, the current count field ($SWU_CUR[n].CURCNT$) contains the cycle count remaining within the current watchpoint period. The SWU decrements this value every cycle until the count reaches zero. At that point, the SWU reloads the $SWU_CUR[n].CURCNT$ field from $SWU_CNT[n]$ register's $SWU_CNT[n].COUNT$ field. In bandwidth mode, the current bandwidth field ($SWU_CUR[n].CURBW$) contains the count of watchpoint matches (bandwidth) accumulated in the current watchpoint period.

In watchpoint mode, the current count field ($SWU_CUR[n].CURCNT$) contains the watchpoint match count remaining within the current watchpoint period. The SWU decrements this value with every watchpoint match until the count reaches zero. At that point, the SWU reloads the $SWU_CUR[n].CURCNT$ field from $SWU_CNT[n]$ register's $SWU_CNT[n].COUNT$ field if the $SWU_CTL[n]$ register's $SWU_CTL[n].CNTRPTEN$ bit is set (=1). In watchpoint mode, the current bandwidth field ($SWU_CUR[n].CURBW$) is undefined.

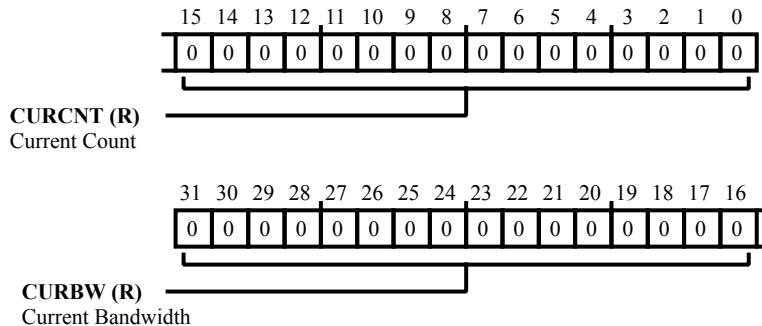


Figure 42-5: SWU_CUR[n] Register Diagram

Table 42-5: SWU_CUR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	CURBW	Current Bandwidth.
15:0 (R/NW)	CURCNT	Current Count.

Global Control Register

The SWU global control register (`SWU_GCTL`) provides SWU reset and enable.

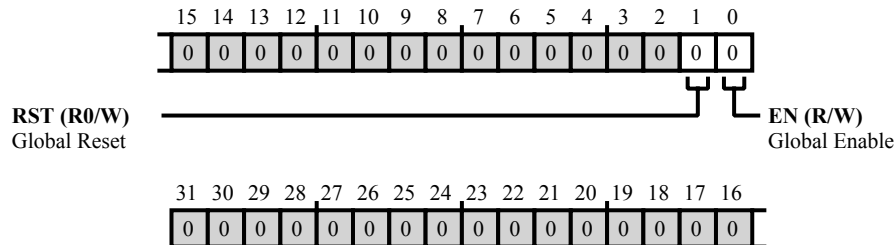


Figure 42-6: SWU_GCTL Register Diagram

Table 42-6: SWU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R0/W)	RST	Global Reset. The <code>SWU_GCTL.RST</code> is write-1-action/read zero and controls the SWU operational state. Setting <code>SWU_GCTL.RST</code> resets all SWU registers to their default values and halts all SWU operations.
		0 No Action
		1 Reset
0 (R/W)	EN	Global Enable. The <code>SWU_GCTL.EN</code> controls the SWU operational state. Clearing <code>SWU_GCTL.EN</code> halts the execution of all watchpoint and bandwidth tracking operations without resetting status registers or associated signals. Setting <code>SWU_GCTL.EN</code> enables the SWU to begin/resume operation with the current configuration and status.
		0 Disable
		1 Enable

Global Status Register

The SWU global status register (`SWU_GSTAT`) contains status bits for all four watchpoint groups.

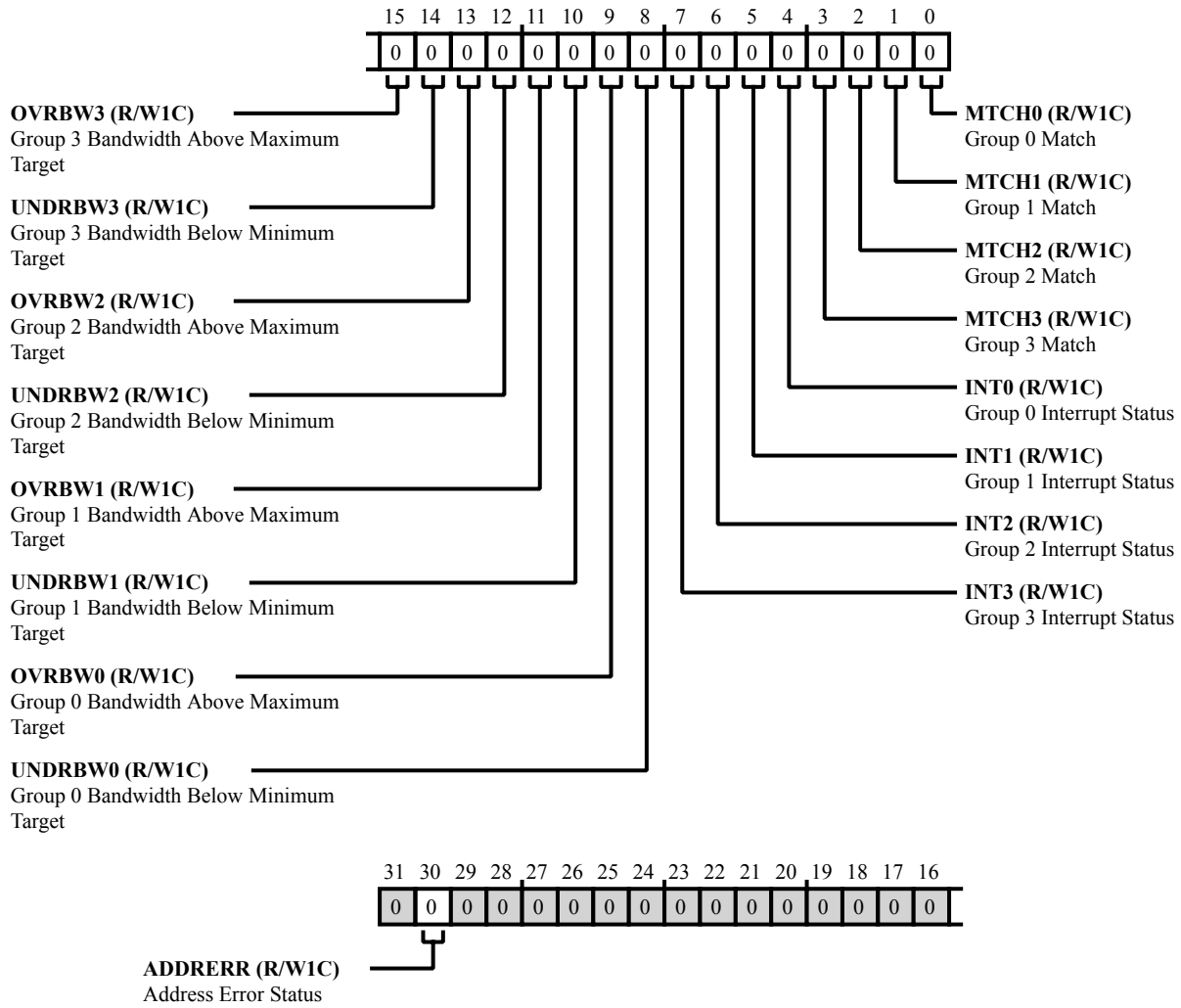


Figure 42-7: SWU_GSTAT Register Diagram

Table 42-7: SWU_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	ADDRERR	Address Error Status. The <code>SWU_GSTAT.ADDRERR</code> indicates that the SWU generated an address error. This status bit is sticky; write-1-to-clear it.
		0 Inactive
		1 Active

Table 42-7: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	OVRBW3	Group 3 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.
		0 Group 3 was not above maximum bandwidth
		1 Group 3 was above maximum bandwidth
14 (R/W1C)	UNDRBW3	Group 3 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.
		0 Group 3 was not below minimum bandwidth
		1 Group 3 was below minimum bandwidth
13 (R/W1C)	OVRBW2	Group 2 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.
		0 Group 2 was not above maximum bandwidth
		1 Group 2 was above maximum bandwidth
12 (R/W1C)	UNDRBW2	Group 2 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.
		0 Group 2 was not below minimum bandwidth
		1 Group 2 was below minimum bandwidth
11 (R/W1C)	OVRBW1	Group 1 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.
		0 Group 1 was not above maximum bandwidth
		1 Group 1 was above maximum bandwidth
10 (R/W1C)	UNDRBW1	Group 1 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.
		0 Group 1 was not below minimum bandwidth
		1 Group 1 was below minimum bandwidth
9 (R/W1C)	OVRBW0	Group 0 Bandwidth Above Maximum Target. The SWU_GSTAT.OVRBW0 - SWU_GSTAT.OVRBW3 -- Group 0 through 3 watch-point bandwidth over maximum target bits. Each maximum bandwidth bit indicate (for each group)s that the measured bandwidth over the period defined by the SWU_CNT[n] register was over the maximum target. This status bit is sticky; write-1-to-clear it.
		0 Group 0 was not above maximum bandwidth
		1 Group 0 was above maximum bandwidth

Table 42-7: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	UNDRBW0	Group 0 Bandwidth Below Minimum Target. The SWU_GSTAT.UNDRBW0 - SWU_GSTAT.UNDRBW3 -- Group 0 through 3 watchpoint bandwidth below minimum target bits. Each minimum bandwidth bit indicates (for each group) that the measured bandwidth over the period defined by the SWU_CNT[n] register was below the minimum target. This status bit is sticky; write-1-to-clear it.
		0 Group 0 was not below minimum bandwidth
		1 Group 0 was below minimum bandwidth
7 (R/W1C)	INT3	Group 3 Interrupt Status. See SWU_GSTAT.INT0 description.
		0 No Interrupt
		1 Interrupt Occurred
6 (R/W1C)	INT2	Group 2 Interrupt Status. See SWU_GSTAT.INT0 description.
		0 No Interrupt
		1 Interrupt Occurred
5 (R/W1C)	INT1	Group 1 Interrupt Status. See SWU_GSTAT.INT0 description.
		0 No Interrupt
		1 Interrupt Occurred
4 (R/W1C)	INT0	Group 0 Interrupt Status. The SWU_GSTAT.INT0 - SWU_GSTAT.INT3 -- Group 0 through 3 interrupt bits. Each interrupt bit indicates (for each group) whether a watchpoint group is contributing to the SWU's interrupt output. This status bit is sticky; write-1-to-clear it.
		0 No interrupt
		1 Interrupt Occurred
3 (R/W1C)	MTCH3	Group 3 Match. See SWU_GSTAT.MTCH0 description.
		0 No Match
		1 Group 3 Watchpoint Match

Table 42-7: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	MTCH2	Group 2 Match. See SWU_GSTAT.MTCH0 description.
		0 No match
		1 Group 2 Watchpoint Match
1 (R/W1C)	MTCH1	Group 1 Match. See SWU_GSTAT.MTCH0 description.
		0 No match
		1 Group 1 Watchpoint Match
0 (R/W1C)	MTCH0	Group 0 Match. The SWU_GSTAT.MTCH0 - SWU_GSTAT.MTCH3 -- Group 0 through 3 match bits. Each match bit indicates (for each group) whether a watchpoint match has occurred in a SWU watchpoint group, as controlled by the group's related watchpoint control register (SWU_CTL[n]). This status bit is sticky; write-1-to-clear it.
		0 No match
		1 Group 0 Watchpoint Match

Bandwidth History Register n

The SWU bandwidth history registers (`SWU_HIST[n]`) contain data copied from a watchpoint group's current bandwidth value (`SWU_CUR[n]` register, `SWU_CUR[n].CURBW` bits) at the end of the last two watchpoint periods. At the end of each watchpoint period, the SWU copies the previous bandwidth value from the `SWU_HIST[n].BWHIST0` field to the `SWU_HIST[n].BWHIST1` field and copies the new bandwidth value from the `SWU_CUR[n].CURBW` field to the `SWU_HIST[n].BWHIST0` field.

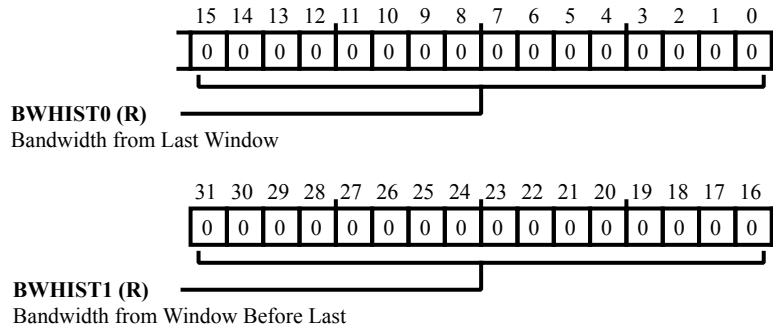


Figure 42-8: `SWU_HIST[n]` Register Diagram

Table 42-8: `SWU_HIST[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	BWHIST1	Bandwidth from Window Before Last.
15:0 (R/NW)	BWHIST0	Bandwidth from Last Window.

ID Register n

The SWU ID registers ($SWU_ID[n]$) contain a 16-bit ID field ($SWU_ID[n].ID$) and a 16-bit ID mask field ($SWU_ID[n].IDMASK$) that watchpoint groups use for ID comparison. The ID on the bus is AND'ed with the $SWU_ID[n].IDMASK$ field, then the watchpoint group compares the result against the $SWU_ID[n].ID$ field.

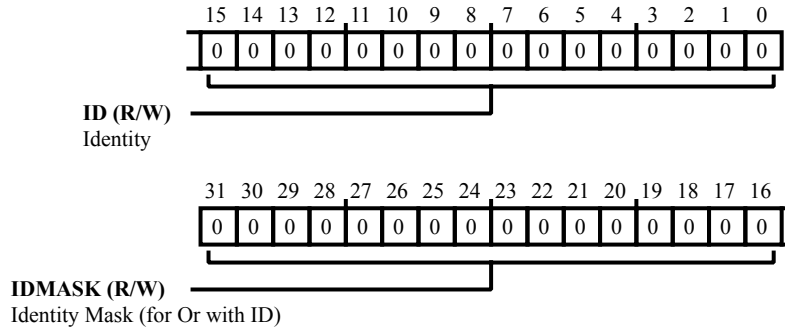


Figure 42-9: SWU_ID[n] Register Diagram

Table 42-9: SWU_ID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	IDMASK	Identity Mask (for Or with ID).
15:0 (R/W)	ID	Identity.

Lower Address Register n

The SWU lower address registers ($SWU_LA[n]$) contain each watchpoint group's lower address for address match comparison. In exact match on $SWU_LA[n]$ address mode ($SWU_CTL[n].ACMPM$ bits =01), the watchpoint group uses only this address for match comparison.

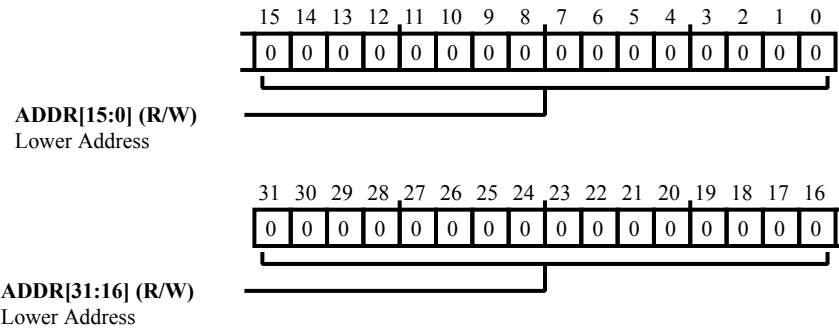


Figure 42-10: SWU_LA[n] Register Diagram

Table 42-10: SWU_LA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Lower Address.

Target Register n

The SWU target registers (`SWU_TARG[n]`) contain a minimum value field (`SWU_TARG[n].BWMIN`) and maximum value field (`SWU_TARG[n].BWMAX`) of bandwidth targets used by watchpoint groups in bandwidth mode. When the bandwidth period expires, if the current bandwidth value (`SWU_CUR[n]` register, `SWU_CUR[n].CURBW` bits) is below the minimum target or above the maximum target, the watchpoint group takes action as enabled by the `SWU_CTL[n]` register's `SWU_CTL[n].MINACT` or `SWU_CTL[n].MAXACT` bits.

In bandwidth mode, note that the watchpoint group increments its count of either data bus transactions or address bus transactions (bursts) as selected by the `SWU_CTL[n].BLENINC` bit. Keep this mode selection in mind when programming the bandwidth target values.

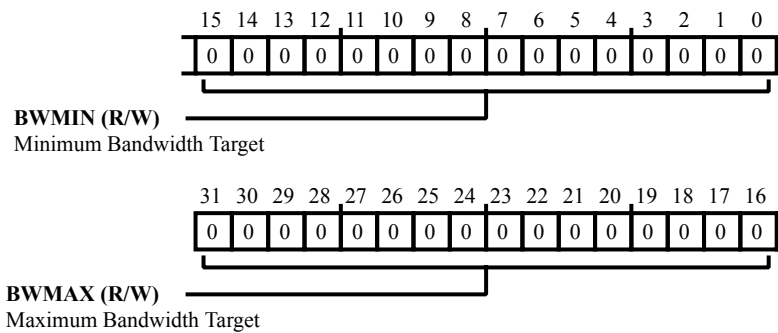


Figure 42-11: `SWU_TARG[n]` Register Diagram

Table 42-11: `SWU_TARG[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	BWMAX	Maximum Bandwidth Target.
15:0 (R/W)	BWMIN	Minimum Bandwidth Target.

Upper Address Register n

The SWU upper address registers ($SWU_UA[n]$) contain each watchpoint group's upper address for address match comparison. In exact match on $SWU_LA[n]$ address mode ($SWU_CTL[n].ACMPM$ bits =01), the $SWU_UA[n]$ is not used for match comparison.

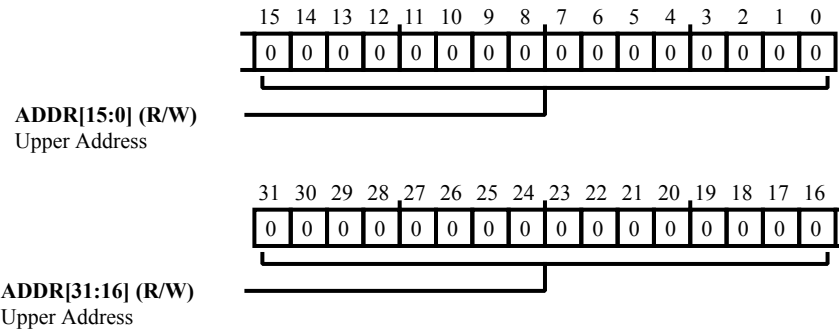


Figure 42-12: SWU_UA[n] Register Diagram

Table 42-12: SWU_UA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

43 Memory Error Protection Unit (MEPU)

Memory Error Protection Unit (MEPU) handles single-bit and double-bit memory error detection and correction across the memories in the chip and controls routing of their interrupts/triggers.

It consists of following blocks:

- Memory Error Controller (MEC)
- Parity Controller (PCTL)

Memory Error Controller (MEC)

The Memory Error Controller (MEC) manages memory parity, ECC errors and warning inputs from various cores and peripherals and sends out interrupt and trigger outputs. It is a generic N x M multiplexer and interrupt controller for error or warning inputs to interrupt and trigger outputs. It has control features such as enable and disable and interrupt masking, and status signaling for input/outputs. For more information on MEC functionality, see the MEC registers description.

MEC Features

The MEC unit has the following features:

- Interrupt controller for memory parity, ECC errors and warnings
- N x M multiplexer for error or warning inputs to interrupt and trigger outputs
- Peripheral bus slave interface for register read and write
- Control and status registers
 - Control for enabling or disabling each input error status and masking its interrupt
 - Control for enabling or disabling each output error interrupt
 - Input error status bits are sticky and write-1-to-clear
 - Output error interrupt status bits are read only
 - Lock feature for control registers write. Status registers capture lock write error.

- Peripheral and component identification registers
- Capability to retain parity, ECC error and warning status even after any reset for fault source debug
- Fully-configurable design

Parity Controller (PCTL)

The Parity Controller (PCTL) handles parity encoding and decoding to and from memory data for single bit memory error detection.

PCTL Features

The PCTL unit has the following features:

- Generic memory parity controller
 - Generation of write parity bits from write data bits
 - Concatenation of parity bits with data bits for memory write data
 - Generation of write enable mask for parity bits to memory
 - Extraction of read data bits from memory read data
 - Detection of read parity error from memory read data
- Even parity logic for parity generation and error detection
- Provision for parity bit interleaving
- Single parity error output per memory instance
- Memory initialization control logic
- Fully-configurable design

MEC Functional Description

The MEC works with the PCTL to detect and correct memory error across the memories in the chip. It also controls the routing of the generated interrupts and triggers.

ADSP-2156x MEC Register List

The Memory Error Controller (MEC) manages memory parity/ecc errors/warning inputs from various cores and peripherals and sends out interrupt/trigger outputs. It is generic N x M multiplexer and interrupt controller for error/warning inputs to interrupt/trigger outputs. It has control features such as enable/disable and interrupt masking, and status signaling for inputs/outputs. For more information on MEC functionality, see the MEC register descriptions.

Table 43-1: ADSP-2156x MEC Register List

Name	Description
MEC_CID0	Component ID0 Register
MEC_CID1	Component ID1 Register
MEC_CID2	Component ID2 Register
MEC_CID3	Component ID3 Register
MEC_CLR	Clear Register
MEC_ECCERR_CTL[y]	ECC Error Control Register
MEC_ECCERR_IMASK[y]	ECC Error Interrupt Mask Register
MEC_ECCERR_STAT[y]	ECC Error Status Register
MEC_EEIRQ_GCTL[q]	ECC Error Interrupt Request Global Control Register
MEC_EEIRQ_GSTAT[q]	ECC Error Interrupt Request Global Status Register
MEC_PEIRQ_GCTL[p]	Parity Error Interrupt Request Global Control Register
MEC_PEIRQ_GSTAT[p]	Parity Error Interrupt Request Global Status Register
MEC_PERR_CTL	Parity Error Control Register
MEC_PERR_IMASK	Parity Error Interrupt Mask Register
MEC_PERR_STAT	Parity Error Status Register
MEC_PID0	Peripheral ID0 Register
MEC_PID1	Peripheral ID1 Register
MEC_PID2	Peripheral ID2 Register
MEC_PID3	Peripheral ID3 Register
MEC_PID4	Peripheral ID4 Register
MEC_PID5	Peripheral ID5 Register
MEC_PID6	Peripheral ID6 Register
MEC_PID7	Peripheral ID7 Register

ADSP-2156x MEC Interrupt List

Table 43-2: ADSP-2156x MEC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
12	MEC0_EEIRQ0	MEC0 ECC Error Interrupt Request		
13	MEC0_EWIRQ0	MEC0 ECC Warning Interrupt Request		
14	MEC0_PEIRQ0	MEC0 Parity Error Interrupt Request	Level	

Table 43-2: ADSP-2156x MEC Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
15	MEC0_PEIRQ1	MEC0 Parity Error Interrupt Request	Level	

ADSP-2156x MEC Trigger List

Table 43-3: ADSP-2156x MEC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
95	MEC0_EEIRQ0	MEC0 ECC Error Interrupt Request	
96	MEC0_PEIRQ0	MEC0 Parity Error Interrupt Request	Level
97	MEC0_PEIRQ1	MEC0 Parity Error Interrupt Request	Level

Table 43-4: ADSP-2156x MEC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

MEPU Block Diagram

The *MEPU Block Diagram* shows the functional blocks within the Memory Error Protection Unit (MEPU) unit.

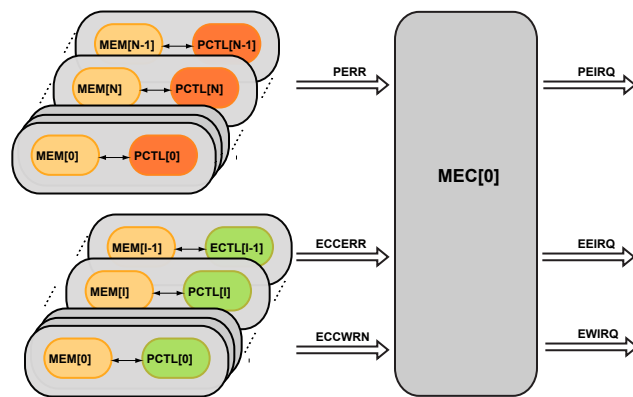


Figure 43-1: MEPU Block Diagram

MEC Block Diagram

The *MEC Interface Block Diagram* shows the functional blocks within the MEC module.

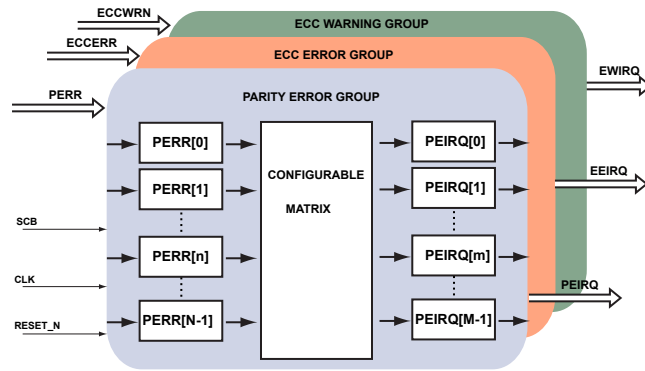


Figure 43-2: MEC Unit Block Diagram

PCTL Block Diagram

The *PCTL Block Diagram* shows the functional blocks within the PCTL module.

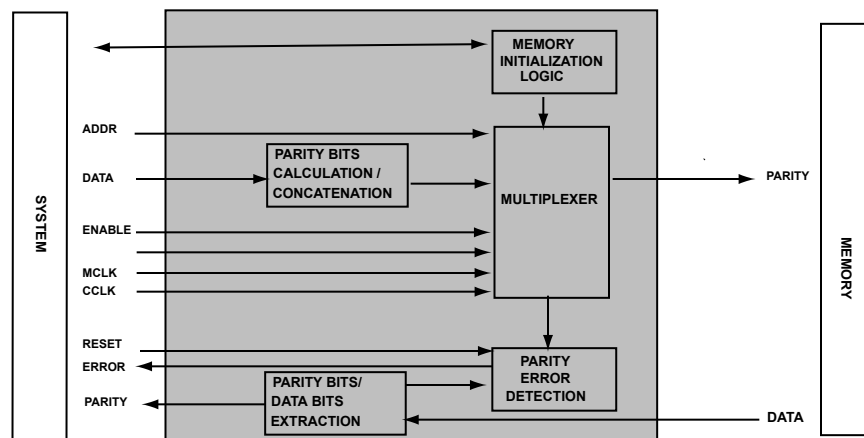


Figure 43-3: PCTL Block Diagram

MEC Architectural Concepts

The MEC unit communicates with the Cortex-A5 directly through the subsystem's SYS bus crossbar. It services MMR addresses associated with the MEC function unit and translates the addresses into specific transcendental MEC functions. The results are read from common result registers shared by all the functions in single-precision floating-point format. The block stalls all read-accesses until the result is ready.

MEC Configuration

The following tables describe the range of values for configuration and the mapping of parity errors.

Table 43-5: Mapping of Input Parity Errors (PERR)

PERR ID	Name	Description
0	PERR_C0_L1CR	Core0 (ARM) L1 Cache RAM Parity Error

Table 43-5: Mapping of Input Parity Errors (PERR) (Continued)

PERR ID	Name	Description
1	PERR_C0_L2CR	Core0 (ARM) L2 Cache RAM Parity Error
2	PERR_C1_L1R	Core1 (SHARC0) L1 RAM Parity Error
3	PERR_C1_L1CR	Core1 (SHARC0) L1 Cache RAM Parity Error
4	PERR_C1_BPR	Core1 (SHARC0) Branch Predictor RAM Parity Error
5	PERR_C2_L1R	Core2 (SHARC1) L1 RAM Parity Error
6	PERR_C2_L1CR	Core2 (SHARC1) L1 Cache RAM Parity Error
7	PERR_C2_BPR	Core2 (SHARC1) Branch Predictor RAM Parity Error
8	PERR_ASRC0_FR	ASRC0 FIFO RAM Parity Error
9	PERR_ASRC1_FR	ASRC1 FIFO RAM Parity Error
10	PERR_ASRC2_FR	ASRC2 FIFO RAM Parity Error
11	PERR_ASRC3_FR	ASRC3 FIFO RAM Parity Error
12	PERR_IIR0_R	IIR0 RAM Parity Error
13	PERR_FIR0_R	FIR0 RAM Parity Error
14	PERR_USB0_FR	USB0 FIFO RAM Parity Error
15	PERR_CAN0_MBR	CAN0 Mailbox RAM Parity Error
16	PERR_CAN0_AMR	CAN0 Acceptance Mask RAM Parity Error
17	PERR_CAN1_MBR	CAN1 Mailbox RAM Parity Error
18	PERR_CAN1_AMR	CAN1 Acceptance Mask RAM Parity Error
19	PERR_TRNG0_DBR	TRNG0 Data Buffer RAM Parity Error
20	PERR_PKA0_DR	PKA0 Data RAM Parity Error
21	PERR_SPE0_BR	SPE0 Buffer RAM Parity Error
22	PERR_SPE0_AR	SPE0 ARC4 RAM Parity Error
23	PERR_EMAC0_TFR	EMAC0 Transmit FIFO RAM Parity Error
24	PERR_EMAC0_RFR	EMAC0 Receive FIFO RAM Parity Error
25	PERR_MSI0_FR	MSI0 FIFO RAM Parity Error
26	PERR_MLB0_DBR	MLB0 Data Buffer RAM Parity Error
27	PERR_MLB0_CTR	MLB0 Channel Table RAM Parity Error
28	PERR_TMC0_TDR	TMC0 Trace Data RAM Parity Error

Table 43-6: Mapping of Output Parity Errors (PERR)

PERR ID	Name	Description
0	PEIRQ_C0	Core0 (ARM) Parity Error Interrupt

Table 43-6: Mapping of Output Parity Errors (PERR) (Continued)

PERR ID	Name	Description
1	PEIRQ_C1	Core1 (SHARC0) Parity Error Interrupt
2	PEIRQ_C2	Core2 (SHARC1) Parity Error Interrupt
3	PEIRQ_SYS	System Peripheral Parity Error Interrupt

Table 43-7: Mapping of Input ECC Errors (ECCERR)

PERR ID	Name	Description
0	ECCERR_L2CTL0	L2CTL0 ECC Error

Table 43-8: Mapping of Output ECC Errors (EEIRQ)

PERR ID	Name	Description
0	EWIRQ_L2CTL0	L2CTL0 ECC Error Interrupt

PCTL Integration

There is one PCTL instance per port per memory instance (for example, one PCTL instance for single port memory and two PCTL instances for dual port memories).

Memory initialization control logic supported by PCTL is used only for instances attached to Arm L1 cache memories.

An additional software trigger master and initialization trigger slave is provided in TRU for starting memory initialization. A pulse from this trigger slave will start initialization of Arm L1 cache memories. When all the locations of a particular memory instance get initialized, PCTL generates a memory initialization done signal for corresponding memory instance. These signals from all PCTL instances corresponding to Arm L1 cache memories are AND'ed to generate a single memory initialization done output which is connected as an interrupt to SEC/GIC and a trigger master to TRU.

ADSP-2156x MEC Register Descriptions

Memory Error Controller (MEC) contains the following registers.

Table 43-9: ADSP-2156x MEC Register List

Name	Description
MEC_CID0	Component ID0 Register
MEC_CID1	Component ID1 Register
MEC_CID2	Component ID2 Register
MEC_CID3	Component ID3 Register
MEC_CLR	Clear Register

Table 43-9: ADSP-2156x MEC Register List (Continued)

Name	Description
MEC_ECCERR_CTL[y]	ECC Error Control Register
MEC_ECCERR_IMASK[y]	ECC Error Interrupt Mask Register
MEC_ECCERR_STAT[y]	ECC Error Status Register
MEC_EEIRQ_GCTL[q]	ECC Error Interrupt Request Global Control Register
MEC_EEIRQ_GSTAT[q]	ECC Error Interrupt Request Global Status Register
MEC_PEIRQ_GCTL[p]	Parity Error Interrupt Request Global Control Register
MEC_PEIRQ_GSTAT[p]	Parity Error Interrupt Request Global Status Register
MEC_PERR_CTL	Parity Error Control Register
MEC_PERR_IMASK	Parity Error Interrupt Mask Register
MEC_PERR_STAT	Parity Error Status Register
MEC_PID0	Peripheral ID0 Register
MEC_PID1	Peripheral ID1 Register
MEC_PID2	Peripheral ID2 Register
MEC_PID3	Peripheral ID3 Register
MEC_PID4	Peripheral ID4 Register
MEC_PID5	Peripheral ID5 Register
MEC_PID6	Peripheral ID6 Register
MEC_PID7	Peripheral ID7 Register

Component ID0 Register

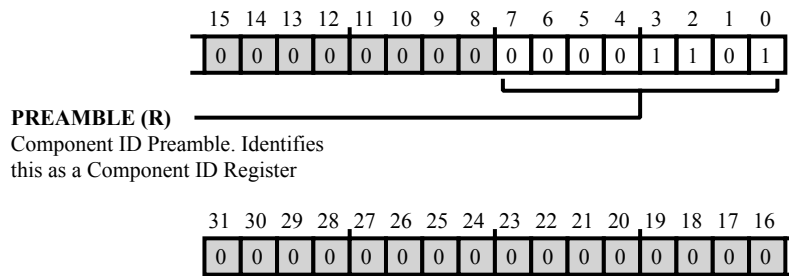


Figure 43-4: MEC_CID0 Register Diagram

Table 43-10: MEC_CID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PREAMBLE	Component ID Preamble. Identifies this as a Component ID Register.

Component ID1 Register

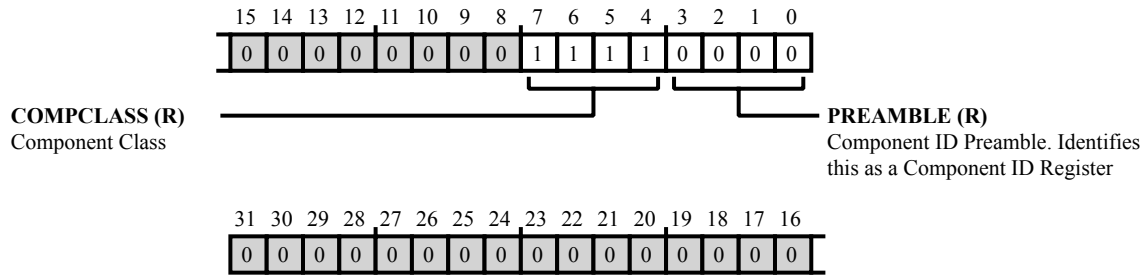


Figure 43-5: MEC_CID1 Register Diagram

Table 43-11: MEC_CID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	COMPCLASS	Component Class. Identifies the Component Class. Dedicated debug blocks (core debug access port, Program Trace, etc.) should identify as CoreSight (i.e. 0x9) and implement the full complement of CoreSight registers including DEVTYPE. All other ADI components should identify as System (i.e. 0xF) components. See CoreSight Architecture Specification for more detail.
		9 CoreSight
		15 System
3:0 (R/NW)	PREAMBLE	Component ID Preamble. Identifies this as a Component ID Register.

Component ID2 Register

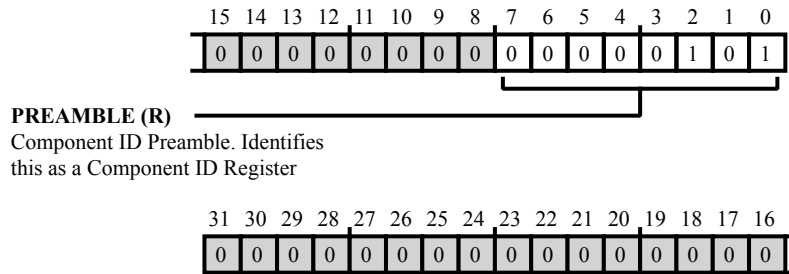


Figure 43-6: MEC_CID2 Register Diagram

Table 43-12: MEC_CID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PREAMBLE	Component ID Preamble. Identifies this as a Component ID Register.

Component ID3 Register

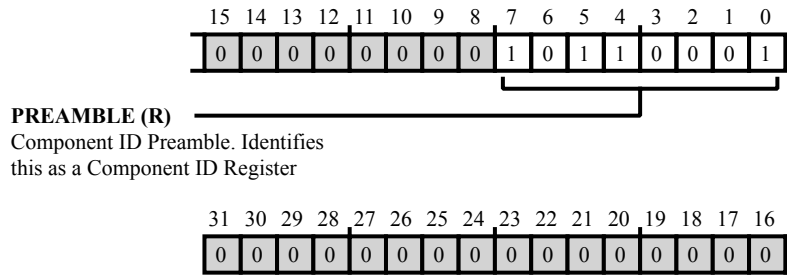


Figure 43-7: MEC_CID3 Register Diagram

Table 43-13: MEC_CID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PREAMBLE	Component ID Preamble. Identifies this as a Component ID Register.

Clear Register

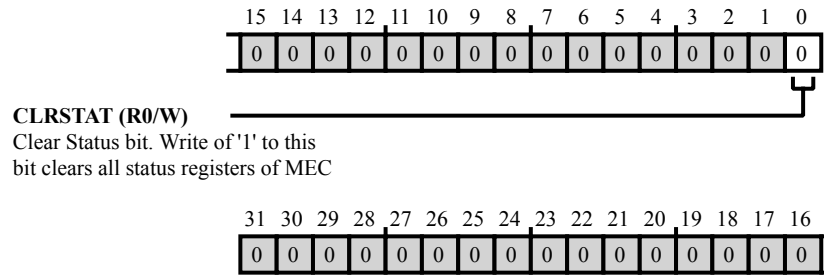


Figure 43-8: MEC_CLR Register Diagram

Table 43-14: MEC_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R0/W)	CLRSTAT	Clear Status bit. Write of '1' to this bit clears all status registers of MEC.

ECC Error Control Register

`MEC_ECCERR_CTL[y]` register bits control enable/disable for ECC error inputs from various cores/peripherals and decide whether their status will be reflected in ECC error status register bits.

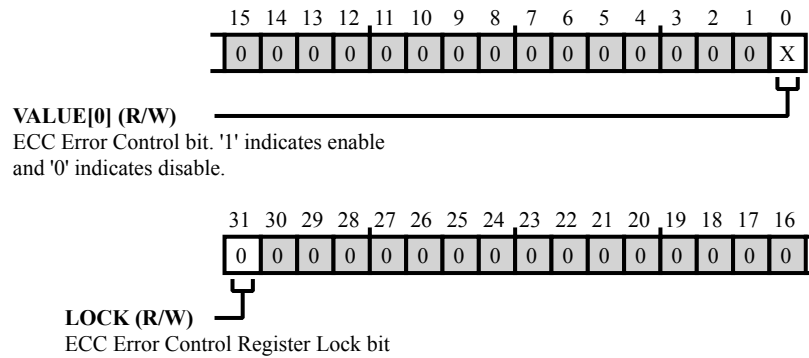


Figure 43-9: MEC_ECCERR_CTL[y] Register Diagram

Table 43-15: MEC_ECCERR_CTL[y] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	ECC Error Control Register Lock bit. If the global lock is enabled (<code>{SPU:CTL:GLCK}</code> bit =1) and the <code>MEC_ECCERR_CTL[y].LOCK</code> bit is enabled, the <code>MEC_ECCERR_CTL[y]</code> register is read only.
		0 Unlock
		1 Lock
0 (R/W)	VALUE	ECC Error Control bit. '1' indicates enable and '0' indicates disable..
		1 L2 CTL ECC Error

ECC Error Interrupt Mask Register

`MEC_ECCERR_IMASK[y]` register bits control interrupt masks for ECC error inputs from various cores/peripherals.

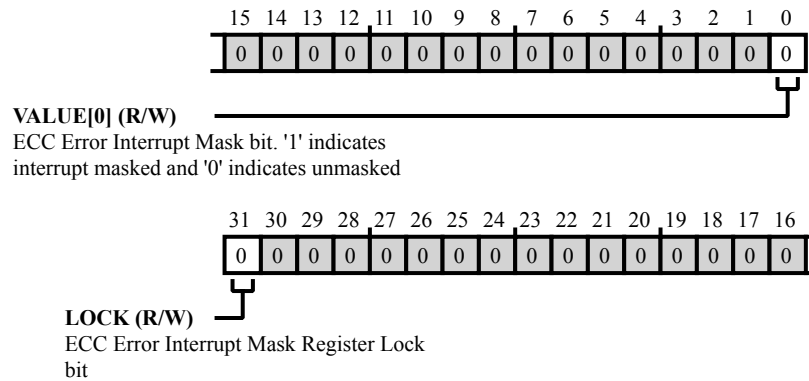


Figure 43-10: MEC_ECCERR_IMASK[y] Register Diagram

Table 43-16: MEC_ECCERR_IMASK[y] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	ECC Error Interrupt Mask Register Lock bit. If the global lock is enabled ({SPU:CTL:GLCK} bit =1) and the <code>MEC_ECCERR_IMASK[y].LOCK</code> bit is enabled, the <code>MEC_ECCERR_IMASK[y]</code> register is read only.
		0 Unlock
		1 Lock
0 (R/W)	VALUE	ECC Error Interrupt Mask bit. '1' indicates interrupt masked and '0' indicates unmasked.
		1 L2 CTL ECC Error

ECC Error Status Register

`MEC_ECCERR_STAT[y]` register bits reflect status for ECC error inputs from various cores/peripherals. Writing '1' to these bits clear corresponding ECC error status.

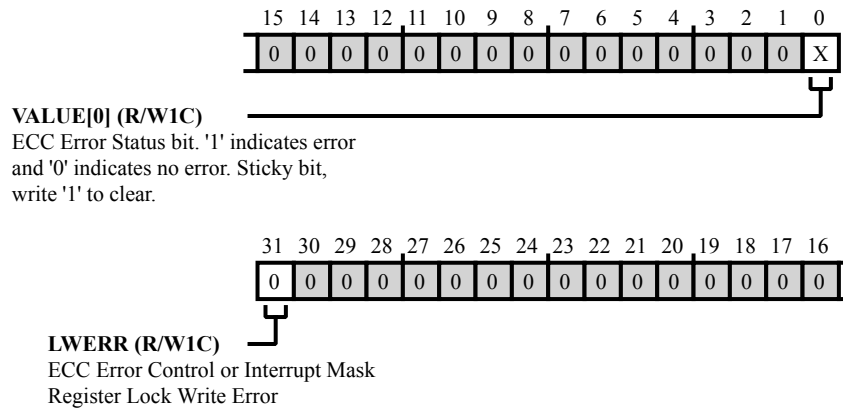


Figure 43-11: MEC_ECCERR_STAT[y] Register Diagram

Table 43-17: MEC_ECCERR_STAT[y] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	ECC Error Control or Interrupt Mask Register Lock Write Error. The <code>MEC_ECCERR_STAT[y].LWERR</code> bit indicates (when set) there was an attempted write to an MEC register while the <code>MEC_ECCERR_CTL[y].LOCK</code> bit was set and while the global lock bit was enabled (<code>{SPU:CTL:GLCK}</code> bit =1). This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
0 (R/W1C)	VALUE	ECC Error Status bit. '1' indicates error and '0' indicates no error. Sticky bit, write '1' to clear..
		1 L2 CTL ECC Error

ECC Error Interrupt Request Global Control Register

`MEC_EEIRQ_GCTL[q]` register bits control enable/disable of ECC error interrupt/trigger outputs.

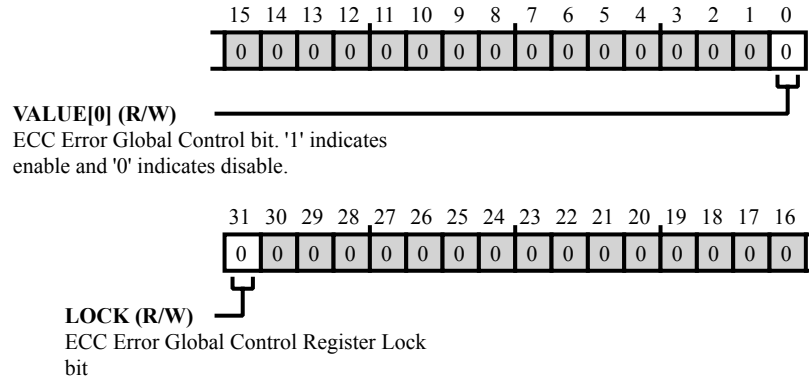


Figure 43-12: MEC_EEIRQ_GCTL[q] Register Diagram

Table 43-18: MEC_EEIRQ_GCTL[q] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	ECC Error Global Control Register Lock bit. If the global lock is enabled (<code>{SPU:CTL:GLCK}</code> bit =1) and the <code>MEC_EEIRQ_GCTL[q].LOCK</code> bit is enabled, the <code>MEC_EEIRQ_GCTL[q]</code> register is read only.
		0 Unlock
		1 Lock
0 (R/W)	VALUE	ECC Error Global Control bit. '1' indicates enable and '0' indicates disable..

ECC Error Interrupt Request Global Status Register

`MEC_EEIRQ_GSTAT[q]` register bits reflect status of ECC error interrupt/trigger outputs.

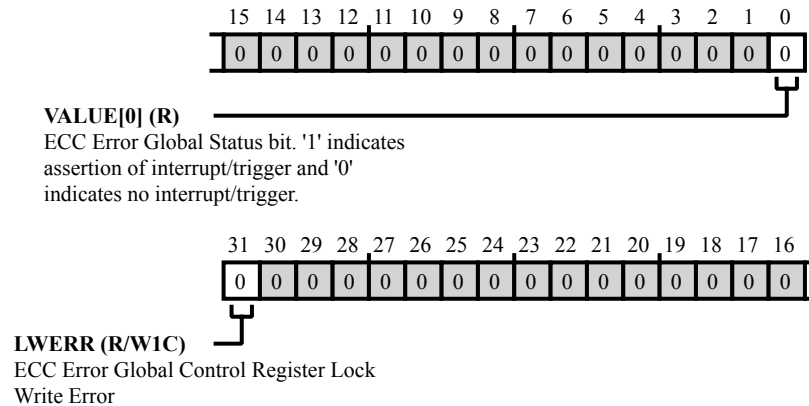


Figure 43-13: MEC_EEIRQ_GSTAT[q] Register Diagram

Table 43-19: MEC_EEIRQ_GSTAT[q] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	ECC Error Global Control Register Lock Write Error. The <code>MEC_EEIRQ_GSTAT[q].LWERR</code> bit indicates (when set) there was an attempted write to an MEC register while the <code>MEC_EEIRQ_GCTL[q].LOCK</code> bit was set and while the global lock bit was enabled (<code>{SPU:CTL:GLCK}</code> bit =1). This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
0 (R/NW)	VALUE	ECC Error Global Status bit. '1' indicates assertion of interrupt/trigger and '0' indicates no interrupt/trigger..

Parity Error Interrupt Request Global Control Register

`MEC_PEIRQ_GCTL[p]` register bits control enable/disable of parity error interrupt/trigger outputs.

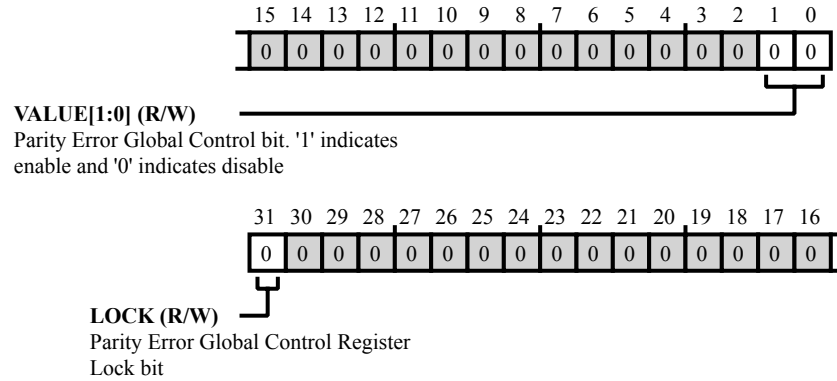


Figure 43-14: MEC_PEIRQ_GCTL[p] Register Diagram

Table 43-20: MEC_PEIRQ_GCTL[p] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Parity Error Global Control Register Lock bit.
		If the global lock is enabled (<code>{SPU:CTL:GLCK}</code> bit =1) and the <code>MEC_PEIRQ_GCTL[p].LOCK</code> bit is enabled, the <code>MEC_PEIRQ_GCTL[p]</code> register is read only.
		0 Unlock
		1 Lock
1:0 (R/W)	VALUE	Parity Error Global Control bit. '1' indicates enable and '0' indicates disable.

Parity Error Interrupt Request Global Status Register

`MEC_PEIRQ_GSTAT[p]` register bits reflect status of parity error interrupt/trigger outputs.

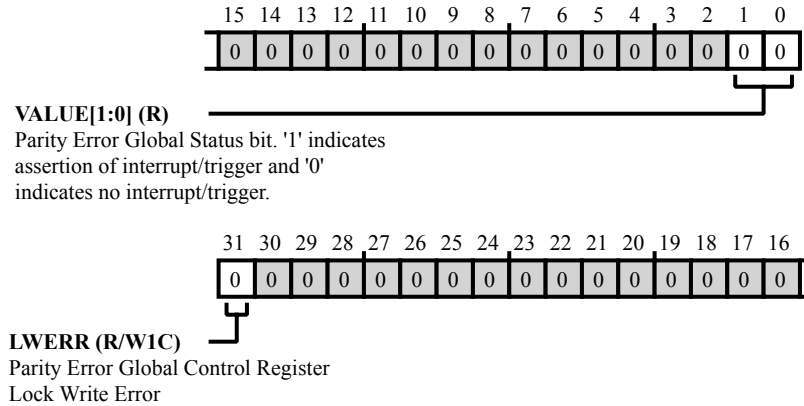


Figure 43-15: `MEC_PEIRQ_GSTAT[p]` Register Diagram

Table 43-21: `MEC_PEIRQ_GSTAT[p]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	Parity Error Global Control Register Lock Write Error. The <code>MEC_PEIRQ_GSTAT[p] . LWERR</code> bit indicates (when set) there was an attempted write to an MEC register while the <code>MEC_PEIRQ_GCTL[p] . LOCK</code> bit was set and while the global lock bit was enabled (<code>{SPU:CTL:GLCK}</code> bit =1). This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
1:0 (R/NW)	VALUE	Parity Error Global Status bit. '1' indicates assertion of interrupt/trigger and '0' indicates no interrupt/trigger..

Parity Error Control Register

`MEC_PERR_CTL` register bits control enable/disable for parity error inputs from various cores/peripherals and decide whether their status will be reflected in parity error status register bits.

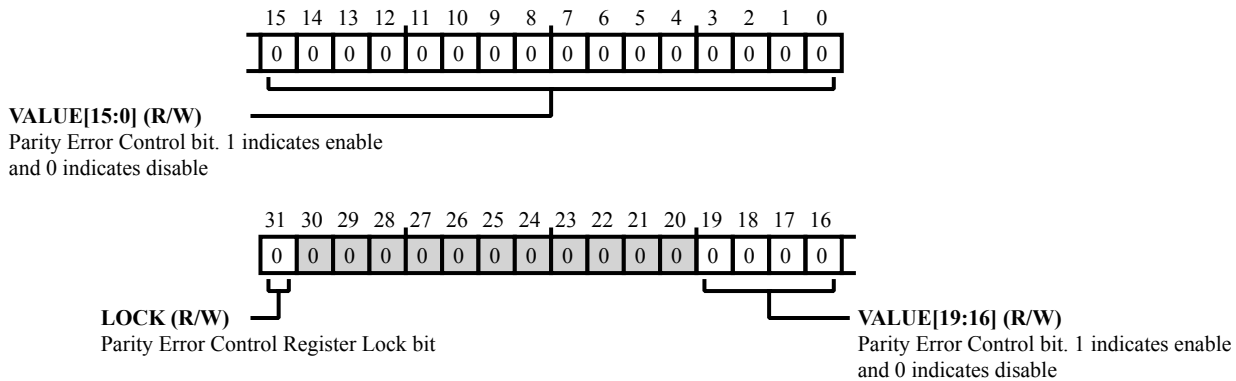


Figure 43-16: MEC_PERR_CTL Register Diagram

Table 43-22: MEC_PERR_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Parity Error Control Register Lock bit.
		If the global lock is enabled (<code>{SPU:CTL:GLCK}</code> bit =1) and the <code>MEC_PERR_CTL.LOCK</code> bit is enabled, the <code>MEC_PERR_CTL</code> register is read only.
		0 Unlock 1 Lock
19:0 (R/W)	VALUE	Parity Error Control bit. 1 indicates enable and 0 indicates disable.
		1 Core 0 L1 RAM Parity Error
		2 Core 0 L1 Cache Parity Error
		4 Branch Predictor Parity Error
		8 ASRC0 FIFO Parity Error
		16 ASRC1 FIFO Parity Error
		32 ASRC2 FIFO Parity Error
		64 ASRC3 FIFO Parity Error
		128 ASRC4 FIFO Parity Error
		256 ASRC5 FIFO Parity Error
		512 ASRC6 FIFO Parity Error
		1024 ASRC7 FIFO Parity Error
2048 IIR0 RAM Parity Error		

Table 43-22: MEC_PERR_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4096	FIR0 RAM Parity Error
		8192	TRNG0 Data Buffer Parity Error
		16384	PKA0 Data RAM Parity Error
		32768	SPE0 Buffer RAM Parity Error
		65536	SPE0 ARC4 RAM Parity Error
		131072	MLB0 Data Buffer RAM Parity Error
		262144	MLB0 Channel Table RAM Parity Error
		524288	TMC0 Trace Data RAM Parity Error

Parity Error Interrupt Mask Register

`MEC_PERR_IMASK` register bits control interrupt masks for parity error inputs from various cores/peripherals.

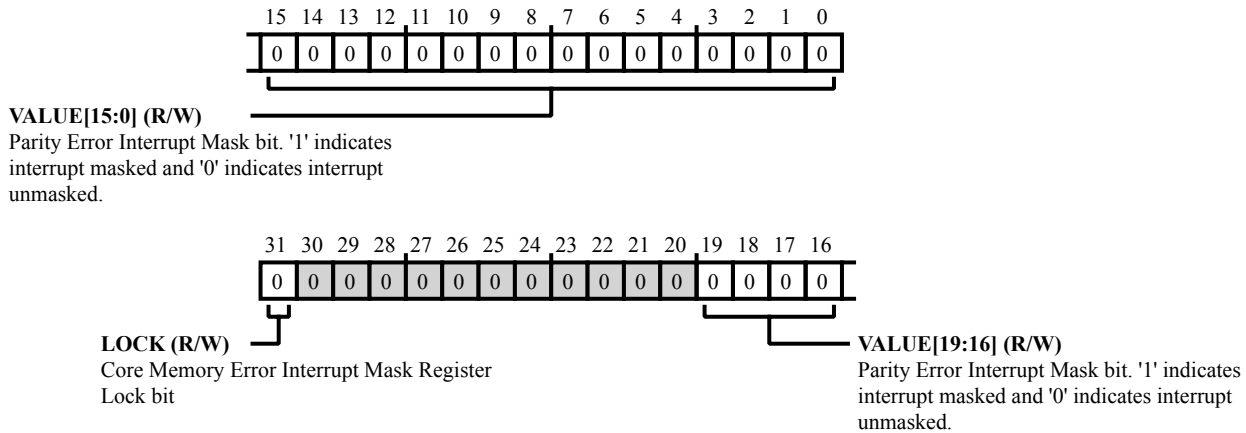


Figure 43-17: MEC_PERR_IMASK Register Diagram

Table 43-23: MEC_PERR_IMASK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Core Memory Error Interrupt Mask Register Lock bit. If the global lock is enabled (<code>{SPU:CTL:GLCK}</code> bit =1) and the <code>MEC_PERR_IMASK</code> . LOCK bit is enabled, the <code>MEC_PERR_IMASK</code> register is read only.
		0 Unlock
		1 Lock
19:0 (R/W)	VALUE	Parity Error Interrupt Mask bit. '1' indicates interrupt masked and '0' indicates interrupt unmasked..
		1 Core 0 L1 RAM Parity Error
		2 Core 0 L1 CACHE Parity Error
		4 Branch Predictor Parity Error
		8 ASRC0 FIFO Parity Error
		16 ASRC1 FIFO Parity Error
		32 ASRC2 FIFO Parity Error
		64 ASRC3 FIFO Parity Error
		128 ASRC4 FIFO Parity Error
		256 ASRC5 FIFO Parity Error
		512 ASRC6 FIFO Parity Error

Table 43-23: MEC_PERR_IMASK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1024	ASRC7 FIFO Parity Error
		2048	IIR0 RAM Parity Error
		4096	FIR0 RAM Parity Error
		8192	TRNG0 Data Buffer Parity Error
		16384	PKA0 Data RAM Parity Error
		32768	SPE0 Buffer RAM Parity Error
		65536	SPE0 ARC4 RAM Parity Error
		131072	MLB0 Data Buffer RAM Parity Error
		262144	MLB0 Channel Table RAM Parity Error
		524288	TMC0 Trace Data RAM Parity Error

Parity Error Status Register

`MEC_PERR_STAT` register bits reflect status for parity error inputs from various cores/peripherals. Writing '1' to these bits clear corresponding parity error status.

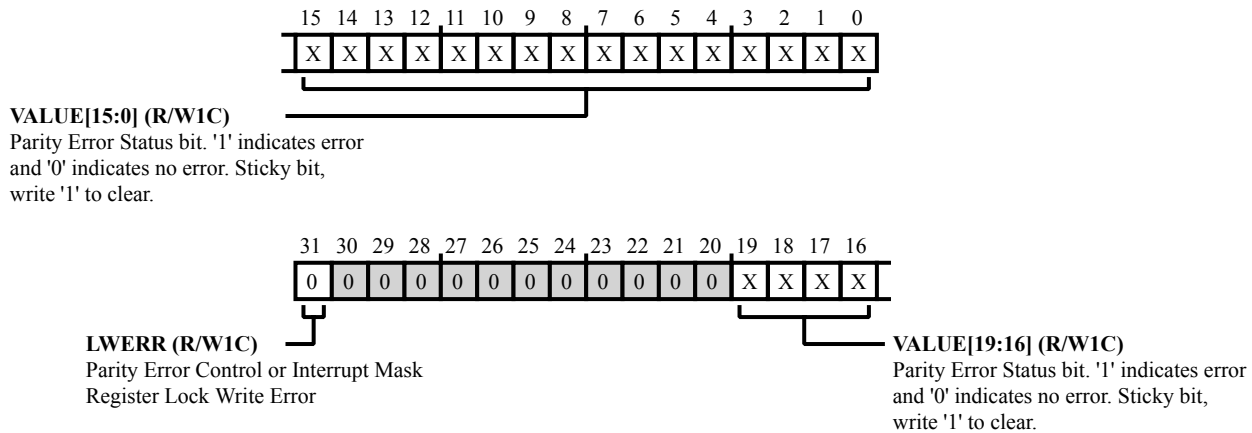


Figure 43-18: MEC_PERR_STAT Register Diagram

Table 43-24: MEC_PERR_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	LWERR	Parity Error Control or Interrupt Mask Register Lock Write Error. The <code>MEC_PERR_STAT.LWERR</code> bit indicates (when set) there was an attempted write to an MEC register while the <code>MEC_PERR_CTL.LOCK</code> bit was set and while the global lock bit was enabled (<code>{SPU:CTL:GLCK}</code> bit =1). This status bit is sticky; write-1-to-clear it.
		0 No Error
		1 Error Occurred
19:0 (R/W1C)	VALUE	Parity Error Status bit. '1' indicates error and '0' indicates no error. Sticky bit, write '1' to clear..
		1 Core 0 L1 RAM Parity Error
		2 Core 0 L1 Cache Parity Error
		4 Branch Predictor Parity Error
		8 ASRC0 FIFO Parity Error
		16 ASRC1 FIFO Parity Error
		32 ASRC2 FIFO Parity Error
		64 ASRC3 FIFO Parity Error
		128 ASRC4 FIFO Parity Error
256 ASRC5 FIFO Parity Error		

Table 43-24: MEC_PERR_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		512	ASRC6 FIFO Parity Error
		1024	ASRC7 FIFO Parity Error
		2048	IIR0 RAM Parity Error
		4096	FIR0 RAM Parity Error
		8192	TRNG0 Data Buffer Parity Error
		16384	PKA0 Data RAM Parity Error
		32768	SPE0 Buffer RAM Parity Error
		65536	SPE0 ARC4 RAM Parity Error
		131072	MLB0 Data Buffer RAM Parity Error
		262144	MLB0 Channel Table RAM Parity Error
		524288	TMC0 Trace Data RAM Parity Error

Peripheral ID0 Register

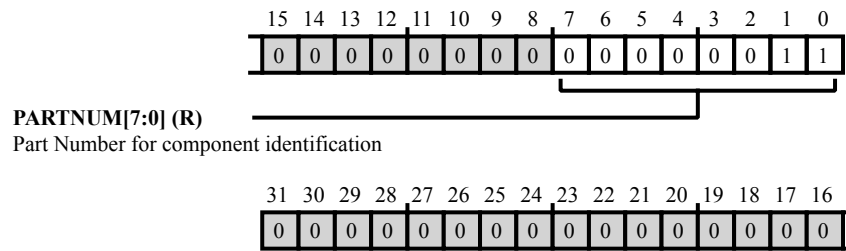


Figure 43-19: MEC_PID0 Register Diagram

Table 43-25: MEC_PID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PARTNUM	Part Number for component identification.

Peripheral ID1 Register

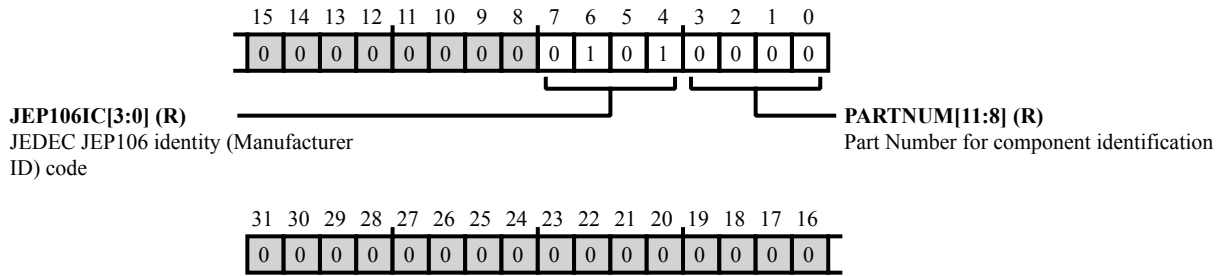


Figure 43-20: MEC_PID1 Register Diagram

Table 43-26: MEC_PID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	JEP106IC	JEDEC JEP106 identity (Manufacturer ID) code.
3:0 (R/NW)	PARTNUM	Part Number for component identification.

Peripheral ID2 Register

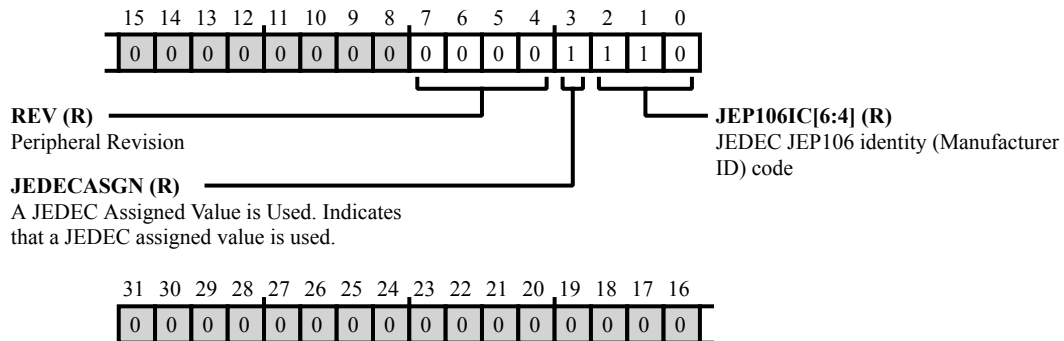


Figure 43-21: MEC_PID2 Register Diagram

Table 43-27: MEC_PID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV	Peripheral Revision.
3 (R/NW)	JEDECASGN	A JEDEC Assigned Value is Used. Indicates that a JEDEC assigned value is used..
2:0 (R/NW)	JEP106IC	JEDEC JEP106 identity (Manufacturer ID) code.

Peripheral ID3 Register

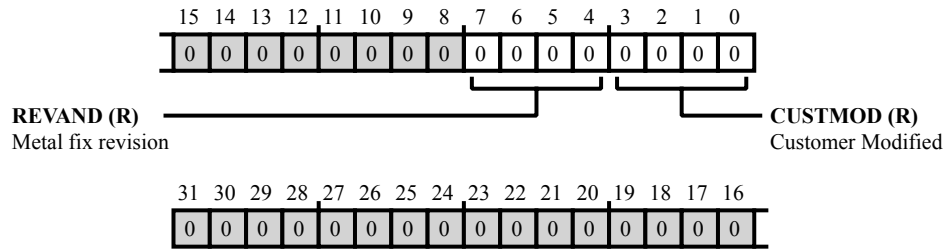


Figure 43-22: MEC_PID3 Register Diagram

Table 43-28: MEC_PID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REVAND	Metal fix revision.
3:0 (R/NW)	CUSTMOD	Customer Modified.

Peripheral ID4 Register

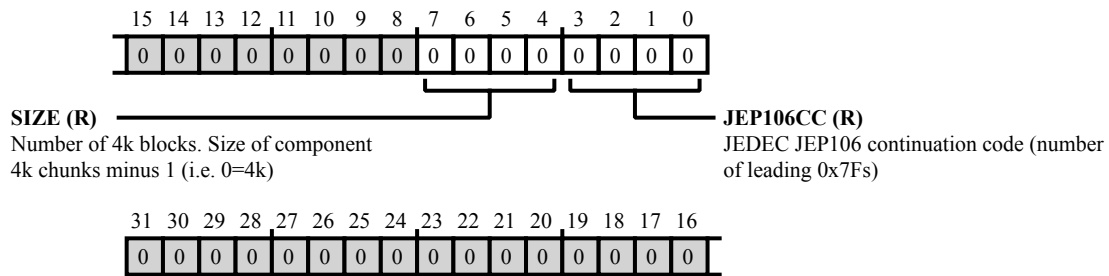


Figure 43-23: MEC_PID4 Register Diagram

Table 43-29: MEC_PID4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	SIZE	Number of 4k blocks. Size of component 4k chunks minus 1 (i.e. 0=4k).
3:0 (R/NW)	JEP106CC	JEDEC JEP106 continuation code (number of leading 0x7Fs).

Peripheral ID5 Register

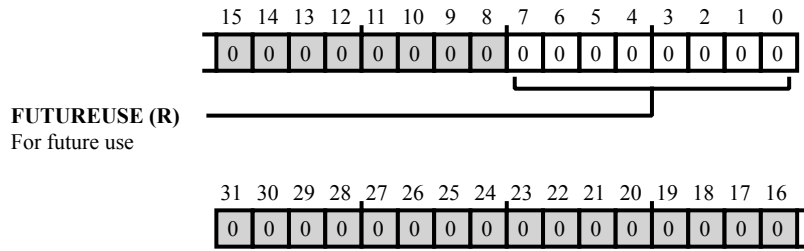


Figure 43-24: MEC_PID5 Register Diagram

Table 43-30: MEC_PID5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	FUTUREUSE	For future use.

Peripheral ID6 Register

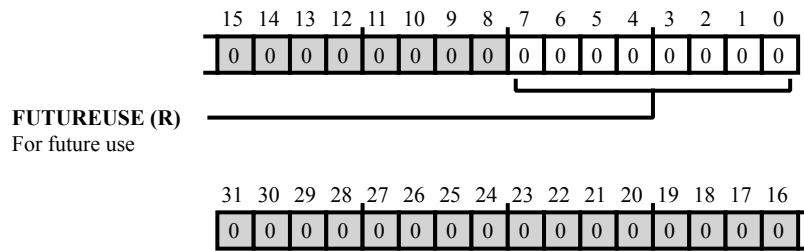


Figure 43-25: MEC_PID6 Register Diagram

Table 43-31: MEC_PID6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	FUTUREUSE	For future use.

Peripheral ID7 Register

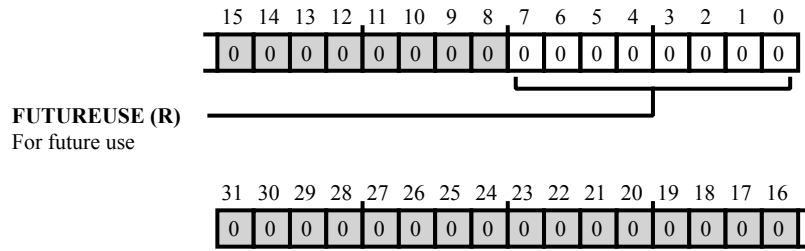


Figure 43-26: MEC_PID7 Register Diagram

Table 43-32: MEC_PID7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	FUTUREUSE	For future use.

44 System Debug and Trace Unit (DBG)

The system debug and trace unit is based on Arm CoreSight technology. CoreSight is a set of architecture specifications defining debug and trace architecture. The processor uses CoreSight infrastructure to provide industry standard debug and trace capabilities.

<http://infocenter.arm.com/help/>

The applicable documentation for more details about the ArmCoreSight feature includes:

- CoreSight PFT Architecture Specification , ARM IHI 0035B (PFT)
- System Trace Macrocell, Programmers' Model Architecture Specification, ARM IHI 0054A (STM)
- CoreSight Trace Memory Controller, ARM DDI0461B (TMC)
- CoreSight Components Technical Reference Manual, ARM DDI 0314H (TPIU)
- Embedded Cross Trigger Technical Reference Manual, ARM DDI 0291A
- ETM Architecture Specification, ARM IHI0014Q

DBG Features

The system debug and trace unit contains the following features.

- System JTAG TAP controller for system debug features, boundary scan, and public JTAG features
- A debug interface to core, and other system resources
- Direct and run-time access to the memory system and system MMRs
- Direct control over system reset
- Support for debug immediately after reset (boot debug)
- Group halt (debug event immediately halts all specified endpoints)
- Real-time on-chip visibility is made available to all developers, including software developers

DBG Functional Description

The following sections provide functional descriptions of the DBG unit.

ADSP-2156x CSPFT Register List

Table 44-1: ADSP-2156x CSPFT Register List

Name	Description
CSPFT_ACTR[n]	Address Comparator Access Type Register
CSPFT_ACVR[n]	Address Comparator Value Register
CSPFT_AUTHSTATUS	Authentication Status Register
CSPFT_CCER	Configuration Code Extension Register
CSPFT_CID0	Component ID0 Register
CSPFT_CID1	Component ID1 Register
CSPFT_CID2	Component ID2 Register
CSPFT_CID3	Component ID3 Register
CSPFT_CIDCMR	Context ID Comparator Mask Register
CSPFT_CIDCVR[n]	Context ID Comparator Value
CSPFT_CLAIMCLR	Claim Tag Clear Register
CSPFT_CLAIMSET	Claim Tag Set Register
CSPFT_CNTENR[n]	Counter Enable Event Register
CSPFT_CNTRLDEVR[n]	Counter Reload Event Register
CSPFT_CNTRLDVR[n]	Counter Reload Value Register
CSPFT_CNTVR[n]	Counter Value Register
CSPFT_CTL	Main Control Register
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register

Table 44-1: ADSP-2156x CSPFT Register List (Continued)

Name	Description
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

ADSP-2156x TAPC Register List

The Test Access Port Controller (TAPC) provides access to debug features. A set of registers governs TAPC operations. For more information on TAPC functionality, see the TAPC register descriptions.

Table 44-2: ADSP-2156x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control Register
TAPC_IDCODE	IDCODE Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register
TAPC_SDBGKEY1	Secure Debug Key 1 Register
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

DBG Block Diagram

The block diagram is shown below.

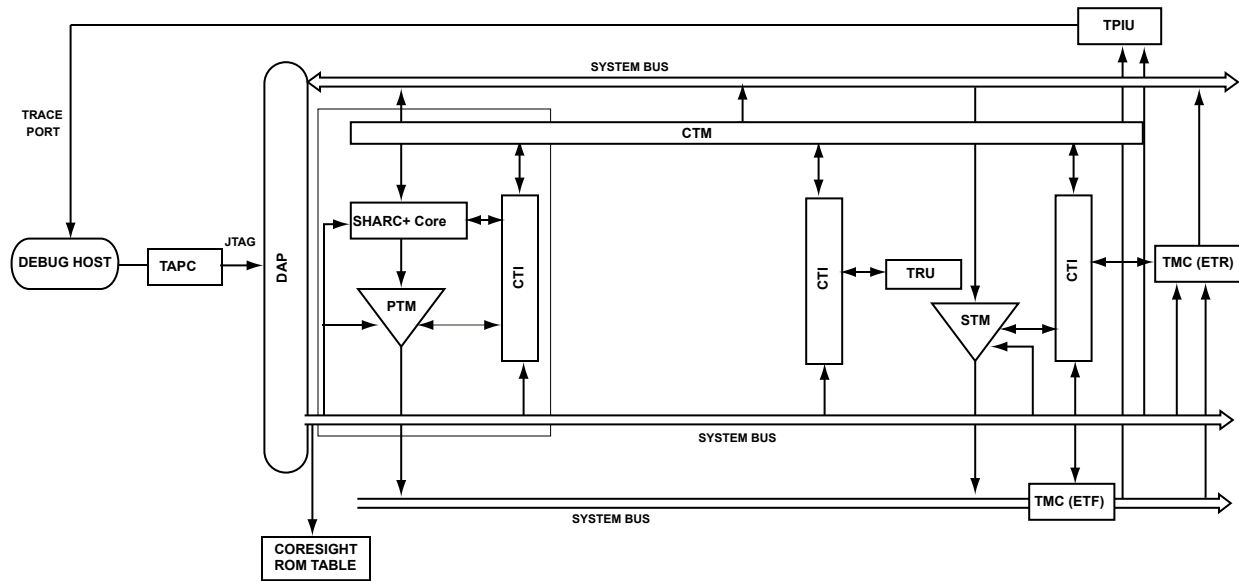


Figure 44-1: ADSP-2156x Block Diagram

DBG Definitions

The following terms are useful when working with the debug features of the processor and programming tools.

Test Access Port Controller (TAPC)

Provides IDCODE and SDBGKEY features.

Debug Access Port (DAP)

Core Sight Interface providing a single port for two debug options: JTAG-DP (JTAG Debug Port), SW-DP (Serial Wire Debug Port)

Embedded Trace Macrocell (ETM)

Provides Arm Core Trace.

Program Trace Macrocell (PTM)

Provides DSP (SHARC +) Core Trace.

Standard Trace Macrocell (STM)

Provides capability to trace up to 32 hardware events and supports 32 software stimulus for data transfer between the user and emulator.

Embedded Cross Trigger (ECT)

The ECTs are responsible managing events and triggers as follows.

- CTI (Cross Trigger Interface) is a CoreSight component for enabling cross triggering of events across a system. From the view of the ECT, it is responsible for combining and mapping trigger requests.
- CTM (Cross Trigger Matrix) is a CoreSight component for connecting multiple CTIs. From the view of the ECT, it is responsible for connecting CTIs and distribution of events.

NOTE: An embedded cross trigger is not the same as the master and slave trigger in the trigger routing unit.

Trace Capture Devices

The trace capture devices capture and format the trace data. There are two trace capture devices in the system:

- ETF - Embedded Trace Funnel - Provides a buffer for burst trace data.
- ETR - Embedded Trace Router - Provides interface for trace data to be stored in system memories.

Trace Port Interface Unit (TPIU)

The TPIU acts as a bridge between the on-chip trace data, with separate IDs, to a data stream. It encapsulates IDs, when required, that the Trace Port Analyzer (TPA) captures.

Test Access Port Controller (TAPC)

TAPC is an independent component daisy-chained to the DAP in the JTAG scan path. The TAPC component provides the product IDCODE (Chip ID) and a security feature.

The TAPC component provides security features to the chip using a debug key match features. This feature permits only those components which have the SDBGKEY to connect and debug the chip. Initially, a 128-bits user key is programmed in the SDBGKEY registers (`TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, `TAPC_SDBGKEY3`) in TAPC by a secure master. The `TAPC_SDBGKEY_CTL.VALID` bit is set.

Then, a user key is entered through the emulator for a match to the initially programmed-entered user key. On a successful match, the chip connects to the emulator. It can be debugged on a failed user key match. Further attempts at keys matching are disabled. JTAG reset or system reset is required to reenables the user key match logic.

Debug Access Ports

The Debug Access Port (DAP) is an implementation of an Arm Debug Interface version 5 (ADIv5) comprising a number of components supplied in a single configuration. All the supplied components fit into the various architectural components for Debug Ports (DPs). The components are used to access the DAP from an external debugger and APs, to access on-chip system resources.

The DAP provides access to all debug and trace capabilities through a single external interface, a JTAG Debug Port (JTAG-DP). The JTAG-DP is based on the IEEE 1149.1 Test Access Port (TAP) and Boundary Scan Architecture. DAP has some additions to support BSCA and IDCODE features that Core Sight DAP does not support.

Trace Unit

The trace module provides instruction, data tracing, and system activity tracing for the processor. The program trace module on the processor is similar to the embedded trace macrocell module provided by the Arm processor. System trace is provided using the system trace macrocell as part of the CoreSight debug and trace interface. The trace module uses an interface based on the AMBA Trace Bus (ATB) standard to output its trace data. The trace data can be either exported to an off-chip trace port analyzer or captured on an on-chip buffer. The PFT and STM modules capture information on the processor both before and after a specific event. The modules add no burden to the processor performance when it runs at full speed.

- [Programmable Flow Trace \(CSPFT\)](#)
- [System Trace Module \(STM\)](#)

Programmable Flow Trace (CSPFT)

When tracing processor execution, trace information can be generated for every instruction the processor executes. This information would be easy to interpret, but would require a prohibitively high trace bandwidth to get the trace data off the chip. With program flow tracing, only branch points are traced. The debugger uses the source code to infer the rest of the executed code.

Certain instructions in the program and events are identified as waypoints. A waypoint is a point where instruction execution involves a change of program flow. The CSPFT only traces those waypoints. These waypoints are:

- All indirect branches
- All direct branches
- Exceptions or interrupts
- Emulator debug entry and exit

When a waypoint occurs, trace data is generated to describe it. From this data and the source code, a trace decompressor can determine what instructions were executed and recreate the instruction flow. To allow the decompressor to calculate where it is in the source code, conditional instructions are marked as waypoints, regardless of whether they pass or fail their condition test. Events like interrupts or debug entry and exit can be promoted from non-waypoint instructions to waypoints to trace the interrupted program flow.

Tracing a waypoint implies the execution of all instructions from the target address of the previous waypoint up to the current waypoint. Non-waypoint instructions are not explicitly traced but the debugger must infer them using the source code. The concept of an instruction block is used throughout this manual and refers to the contiguous block of instructions between two waypoints.

The programming model and function is a subset of the PTM (Programmable Trace Module) of Arm.

System Trace Module (STM)

The STM is a trace source that is integrated into a CoreSight system, and is designed primarily for high-bandwidth trace of instrumentation embedded into software. The STM enables tracing of system activity from various sources:

- Instrumented software, using memory-mapped stimulus ports
- Hardware events

The STM supports the following features:

- Multiple software masters writing software instrumentation independently. Each master can use multiple stimulus ports.
- Time stamping of the system activity. The time stamp is a global time stamp which can be shared with other trace sources in the system to enable correlation of activity from multiple trace sources.
- Indication that specific events have occurred, such as a particular hardware event or a piece of software instrumentation. These events are known as triggers and can be indicated in the trace stream, or through signals to other system components.

Thirty-two hardware event resources are connected as output from the TRU which allows monitoring all of the hardware events that can generate a trigger.

Embedded Cross Trigger (ECT)

ECT provides an interface to the CoreSight debug system enabling the subsystems to interact (cross trigger) with each other. ECT provides a mechanism to forward debug events from one connected subsystem to another connected subsystem. The different subsystems connected to the ECT depend on the processor design. For example, in a multiprocessor system, the interface can be connected to each of the cores and one to the trace subsystem. For a uniprocessor system, the interface can include just the core and trace subsystem connection.

- CTI Cross trigger interface. A CoreSight component for enabling cross triggering of events across a system.
- CTM Cross trigger matrix. A CoreSight component for connecting multiple cross trigger interfaces.

The main function of the ECT (CTI and CTM) is to pass debug events from one connected subsystem to another connected subsystem.

For example, the ECT can communicate debug state information from the core to trace subsystem for a single processor system or to another core in a multiprocessor-based system. Program execution on both the subsystem can be stopped at the same time.

The *Trigger Flow* figure shows a simple debug trigger flow sequence. On each CTI, there are four channel, eight input, and eight output debug triggers. All the eight inputs and outputs can be mapped to a single channel or different channels based on the debug trigger to channel mapping. When a trigger input occurs, it creates a channel event. The channel event causes all the output debug triggers to be triggered. The embedded cross trigger depends on the debug trigger it connects to.

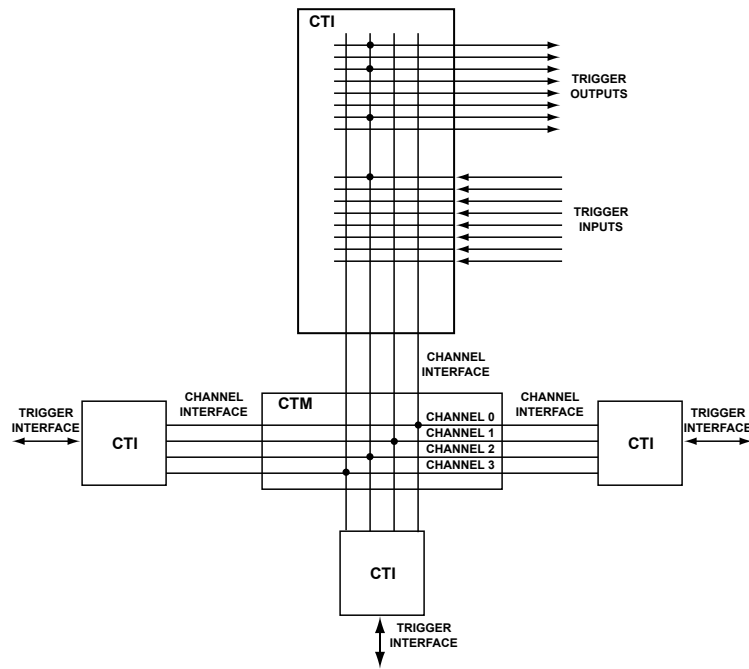


Figure 44-2: Trigger Flow

The *ECT Integration* figure shows 3 CTIs in the system that connect to the core, trace, and system module, respectively. The CTIs all interconnect through the CTM. This configuration allows the core to trigger debug events on the trace, on the system and on the core itself.

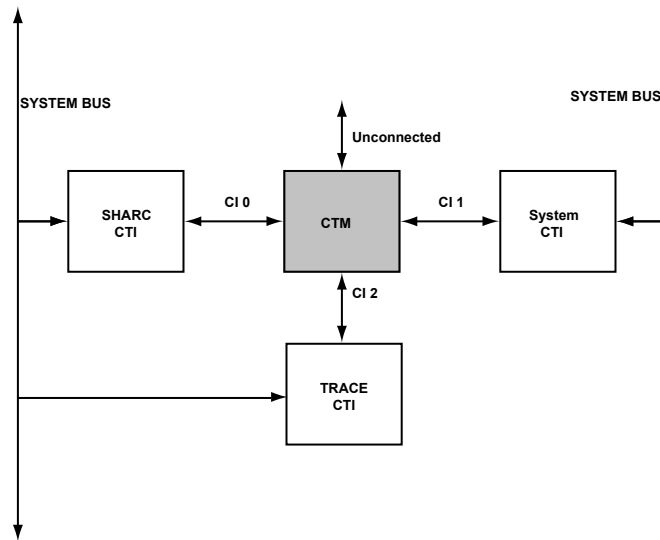


Figure 44-3: ECT Integration

CTI Debug Trigger Tables

The *System CTI Trigger Connection* tables show the debug trigger that connects to each CTI.

Table 44-3: System CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7:3]	From TRU	CTITRIGOUT[7]	To TRU and as DBGRESTART to CTITRIGOUTACK[1] and CTITRIGOUTACK[0]
CTITRIGIN[2]	$\overline{\text{RCU_SYSRSTB}}$	CTITRIGOUT[6:3]	To TRU
CTITRIGIN[1:0]	From TRU	CTITRIGOUT[2]	To RCU as system reset request
		CTITRIGOUT[1]	To TRU and system peripheral halt
		CTITRIGOUT[0]	To TRU and system fabric halt

Table 44-4: Trace CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	STM ASYNCOUT	CTITRIGOUT[7:2]	Unused
CTITRIGIN[6]	STM TRIGOUTHETE	CTITRIGOUT [5]	TPIU FLUSHIN
CTITRIGIN[5]	STM TRIGOUTSW	CTITRIGOUT [4]	TPIU TRIGIN
CTITRIGIN[4]	STM TRIGOUTSPTE	CTITRIGOUT [3]	ETR FLUSHIN
CTITRIGIN[3]	ETR FULL	CTITRIGOUT [2]	ETR TRIGIN
CTITRIGIN[2]	ETR ACQCOMP	CTITRIGOUT [1]	ETF FLUSHIN
CTITRIGIN[1]	ETF FULL	CTITRIGOUT [0]	ETF TRIGIN
CTITRIGIN[0]	ETF ACQCOMP		

Table 44-5: ADSP-2156x CTI Trace Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	Tied Low (Unused)	CTITRIGOUT[7]	DBGRESTART
CTITRIGIN[6]	PTM TRIGGER	CTITRIGOUT[6]	SEC
CTITRIGIN[5:2]	PTM EXTOUT[3:0]	CTITRIGOUT[5]	Unused
CTITRIGIN[1]	Tied Low (Unused)	CTITRIGOUT[4:1]	PTM EXTIN[3:0]
CTITRIGIN[0]	DBGTRIGGER	CTITRIGOUT[0]	EDBGRQ

Table 44-6: Trigger Descriptions

Signal Name	Description
STM ASYNCOUT	Alignment synchronization output. This signal is asserted for one clock cycle when an ASYNC-VERSION-FREQ sequence is completely output on the ATB, and can be used for cross-triggering.

Table 44-6: Trigger Descriptions (Continued)

Signal Name	Description
STM TRIGOUTHETE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMHETER.
STM TRIGOUTSW	Trigger output. This signal is asserted for one clock cycle when a trigger event is generated on writes to a TRIG location in the extended stimulus port registers.
STM TRIGOUTSPTE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMSPTER.
ETR / ETF FULL	This output indicates the value of the full bit in the ETR/ETF status register. A full bit indicates the amount of data in ETF/ETR. An output signal indicating when the circular buffer or FIFO is full, or within a program-mable amount of being full.
ETR / ETF ACQCOMP	This output indicates the value of the FtEmpty bit in the ETR/ETF status register. The bit it is set when trace capture has stopped. An output signal indicating when trace capture has stopped, usually following a trigger condition.
ETR/ETF/ TPIU TRIGIN	This input can cause a trigger event (Start /Stop Trigger). An input signal indicating when a trigger condition has occurred.
ETR/ETF /TPIU FLUSHIN	This input can cause a Trace flush. An input signal indicating a flush request
PTM/ETM TRIGGER	Trigger Input. Trigger event specifies the conditions that must be met to generate a trigger.
PTM/ETM EXTOUT[3:0]	PTM Output.
PTM/ETM EXTIN[3:0]	PTM Input
SEC	CTI Interrupt
DBGRESTART	This is an output from the CTI to a processor core or to system to return from debug mode.
DBGTRIGGER	This is a processor core output signal indicating that the core has moved to debug mode. If the CTIs are setup for synchronous halt, it will generate EDBGGRQ to everyone else.
EDBGRQ	This is an output from CTI and an input (as debug halt) to a processor core and to the system in general. It can assert as a result of another core going to emulation space (DBGTRIGGER) or by setting the corresponding bit in the CTI.
SYS_DBGRESTART	This is an output from the CTI to system to return from debug mode.
To TRU	TRU Master Event
From TRU	TRU Slave Event

ADSP-2156x CSPFT Register Descriptions

Program Flow Trace (CSPFT) contains the following registers.

Table 44-7: ADSP-2156x CSPFT Register List

Name	Description
CSPFT_ACTR[n]	Address Comparator Access Type Register
CSPFT_ACVR[n]	Address Comparator Value Register
CSPFT_AUTHSTATUS	Authentication Status Register
CSPFT_CCER	Configuration Code Extension Register
CSPFT_CID0	Component ID0 Register
CSPFT_CID1	Component ID1 Register
CSPFT_CID2	Component ID2 Register
CSPFT_CID3	Component ID3 Register
CSPFT_CIDCMR	Context ID Comparator Mask Register
CSPFT_CIDCVR[n]	Context ID Comparator Value
CSPFT_CLAIMCLR	Claim Tag Clear Register
CSPFT_CLAIMSET	Claim Tag Set Register
CSPFT_CNTENR[n]	Counter Enable Event Register
CSPFT_CNTRLDEVR[n]	Counter Reload Event Register
CSPFT_CNTRLDVR[n]	Counter Reload Value Register
CSPFT_CNTVR[n]	Counter Value Register
CSPFT_CTL	Main Control Register
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register

Table 44-7: ADSP-2156x CSPFT Register List (Continued)

Name	Description
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

Address Comparator Access Type Register

The `CSPFT_ACTR[n]` register specifies whether the context ID needs to match.

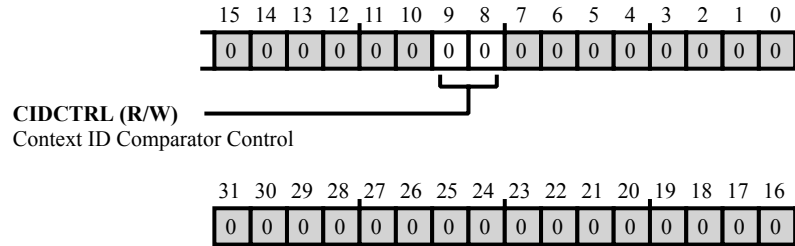


Figure 44-4: `CSPFT_ACTR[n]` Register Diagram

Table 44-8: `CSPFT_ACTR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	CIDCTRL	Context ID Comparator Control. The <code>CSPFT_ACTR[n].CIDCTRL</code> contains the ID comparator control value.
		0 Ignore Context ID
		1 Match if Context ID Comparator 0 Matches
		2 Match if Context ID Comparator 1 Matches
		3 Match if Context ID Comparator 2 Matches

Address Comparator Value Register

The `CSPFT_ACVR[n]` register holds an address for comparison.

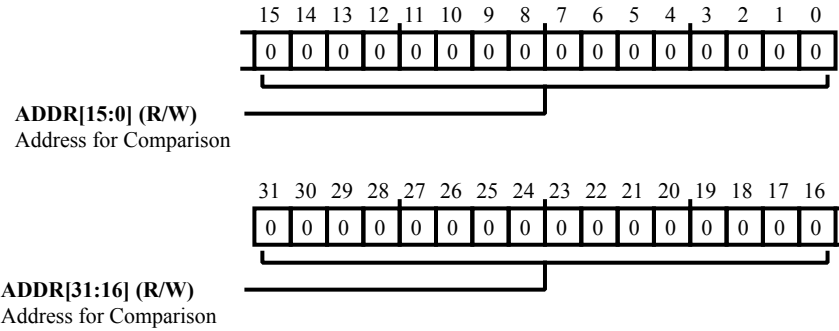


Figure 44-5: `CSPFT_ACVR[n]` Register Diagram

Table 44-9: `CSPFT_ACVR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address for Comparison. The <code>CSPFT_ACVR[n].ADDR</code> bit field contains the address used for comparison.

Authentication Status Register

The `CSPFT_AUTHSTATUS` register reports the level of tracing currently permitted based on the DBGEN signal.

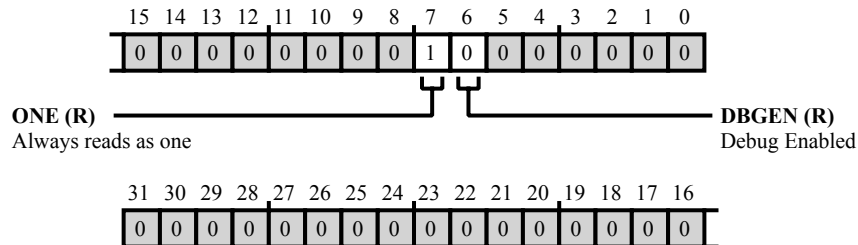


Figure 44-6: CSPFT_AUTHSTATUS Register Diagram

Table 44-10: CSPFT_AUTHSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	ONE	Always reads as one.
6 (R/NW)	DBGEN	Debug Enabled. The <code>CSPFT_AUTHSTATUS</code> . <code>DBGEN</code> bit indicates that invasive debug is enabled. Normally, <code>NIDEN</code> is used in conjunction with a signal that enables invasive debug, <code>DBGEN</code> . Non-invasive debug is disabled only if both <code>NIDEN</code> and <code>DBGEN</code> signals are LOW. In a PTM, typically these signals are ORed together and the result is used to determine whether non-invasive debug is enabled.

Configuration Code Extension Register

The `CSPFT_CCER` register holds extra feature information. (See `CSPFT_HWFEAT`.)

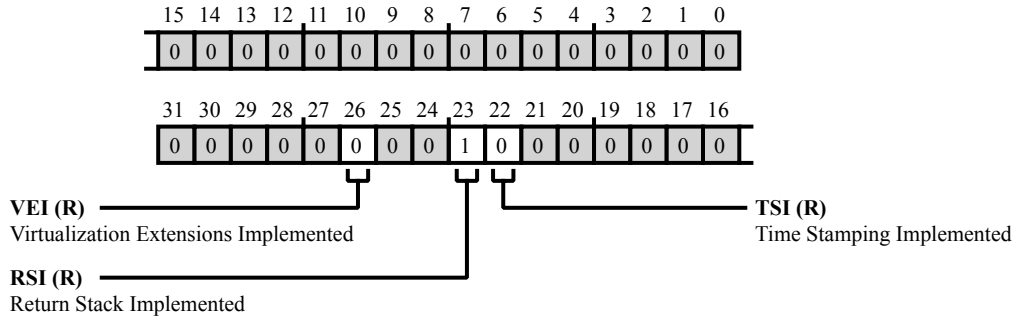


Figure 44-7: CSPFT_CCER Register Diagram

Table 44-11: CSPFT_CCER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/NW)	VEI	Virtualization Extensions Implemented. The <code>CSPFT_CCER.VEI</code> bit indicates if the virtualization extensions are implemented
		0 Not Implemented
		1 Implemented
23 (R/NW)	RSI	Return Stack Implemented. The <code>CSPFT_CCER.RSI</code> bit indicates if a return stack is implemented.
		0 Not Implemented
		1 Implemented
22 (R/NW)	TSI	Time Stamping Implemented. The <code>CSPFT_CCER.TSI</code> bit indicates if time stamping is implemented.
		0 Disabled
		1 Enabled

Component ID0 Register

The `CSPFT_CID0` register holds sections of the CoreSight Component ID for CSPFT.

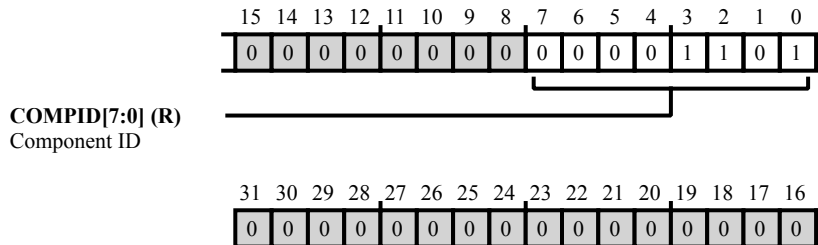


Figure 44-8: CSPFT_CID0 Register Diagram

Table 44-12: CSPFT_CID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The <code>CSPFT_CID0.COMPID</code> bit field identifies this component as a CoreSight component.

Component ID1 Register

The `CSPFT_CID1` register holds sections of the CoreSight Component ID for CSPFT.

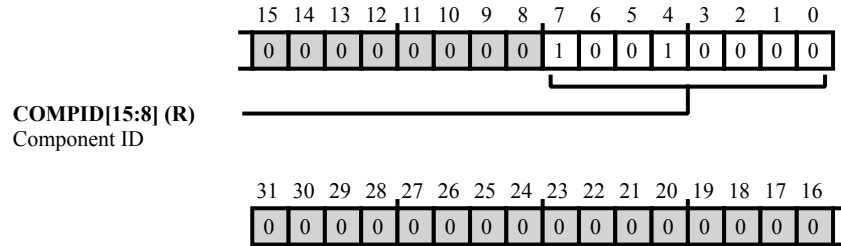


Figure 44-9: CSPFT_CID1 Register Diagram

Table 44-13: CSPFT_CID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The <code>CSPFT_CID1.COMPID</code> bit field identifies this component as a CoreSight component.

Component ID2 Register

The `CSPFT_CID2` register holds sections of the CoreSight Component ID for CSPFT.

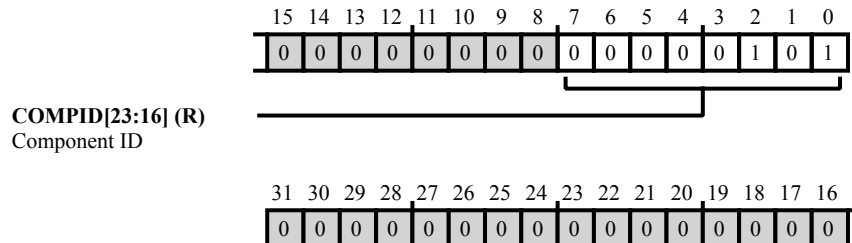


Figure 44-10: CSPFT_CID2 Register Diagram

Table 44-14: CSPFT_CID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The <code>CSPFT_CID2.COMPID</code> bit field identifies this component as a CoreSight component.

Component ID3 Register

The `CSPFT_CID3` register holds sections of the CoreSight Component ID for CSPFT.

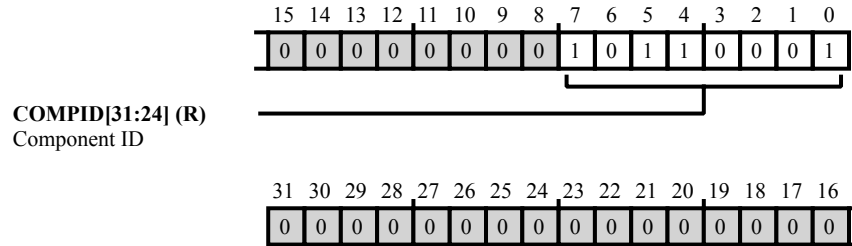


Figure 44-11: CSPFT_CID3 Register Diagram

Table 44-15: CSPFT_CID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The <code>CSPFT_CID3.COMPID</code> bit field identifies this component as a CoreSight component.

Context ID Comparator Mask Register

The `CSPFT_CIDCMR` register holds a 32-bit mask for use for all context ID comparisons.

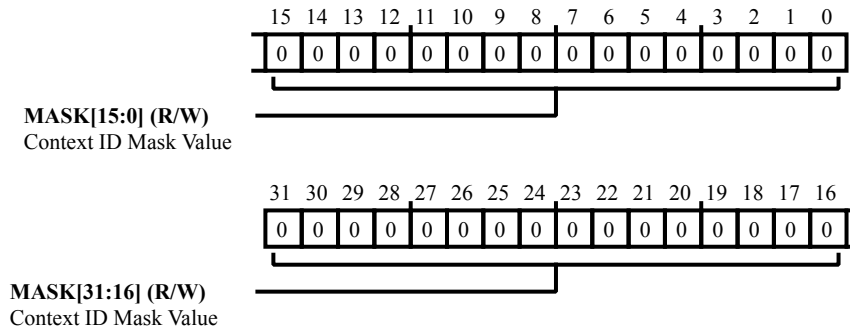


Figure 44-12: `CSPFT_CIDCMR` Register Diagram

Table 44-16: `CSPFT_CIDCMR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MASK	Context ID Mask Value. The <code>CSPFT_CIDCMR.MASK</code> bit field holds a 32-bit mask for use in all context ID comparisons.

Context ID Comparator Value

The `CSPFT_CIDCVR[n]` register holds a context ID value for comparison.

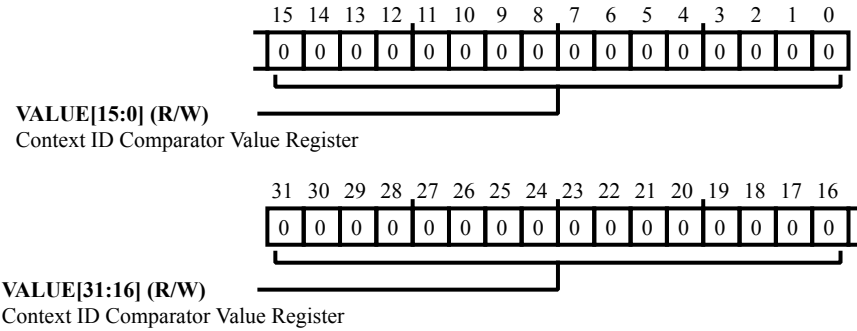


Figure 44-13: `CSPFT_CIDCVR[n]` Register Diagram

Table 44-17: `CSPFT_CIDCVR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Context ID Comparator Value Register. The <code>CSPFT_CIDCVR[n].VALUE</code> bit field holds a context ID value for comparison.

Claim Tag Clear Register

The `CSPFT_CLAIMCLR` register is used to clear bits in the claim tag or get the current value of the claim tag.

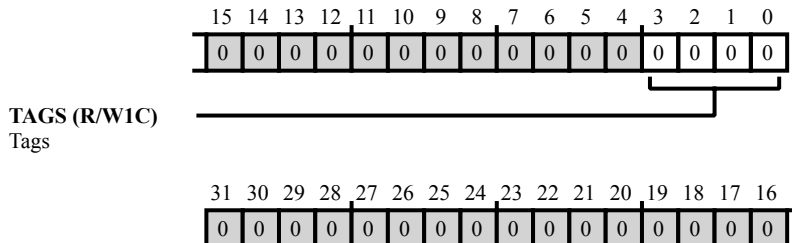


Figure 44-14: CSPFT_CLAIMCLR Register Diagram

Table 44-18: CSPFT_CLAIMCLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1C)	TAGS	Tags. A read of the <code>CSPFT_CLAIMCLR.TAGS</code> bit field returns the current value, a write clears bits.

Claim Tag Set Register

The `CSPFT_CLAIMSET` register is used to set bits in the claim tag and find the number of bits supported by the claim tag.

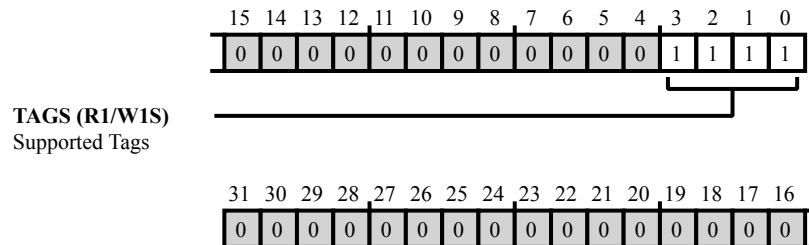


Figure 44-15: `CSPFT_CLAIMSET` Register Diagram

Table 44-19: `CSPFT_CLAIMSET` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R1/W1S)	TAGS	Supported Tags. The <code>CSPFT_CLAIMSET.TAGS</code> bit field sets bits in the claim tag and finds the number of bits supported by the claim tag.

Counter Enable Event Register

The `CSPFT_CNTENR[n]` register describes the event that enables the corresponding counter.

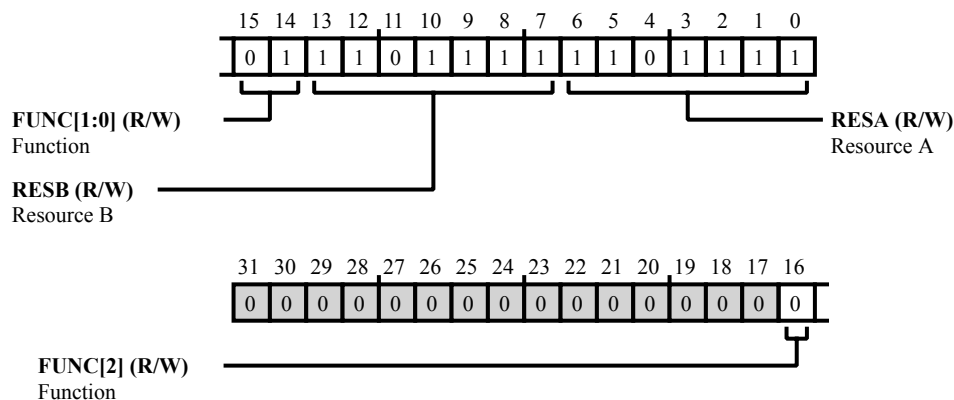


Figure 44-16: `CSPFT_CNTENR[n]` Register Diagram

Table 44-20: `CSPFT_CNTENR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:14 (R/W)	FUNC	Function. The <code>CSPFT_CNTENR[n].FUNC</code> bit field specifies the logical operation that combines the two resources that define the event.
		0 A
		1 NOT(A)
		2 A AND B
		3 NOT(A) AND B
		4 NOT(A) AND NOT(B)
		5 A OR B
		6 NOT(A) OR B
		7 NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The <code>CSPFT_CNTENR[n].RESB</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_CNTENR[n].FUNC</code> field (See <code>CSPFT_CNTENR[n].RESA</code> for list of possible values).
6:0 (R/W)	RESA	Resource A. The <code>CSPFT_CNTENR[n].RESA</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_CNTENR[n].FUNC</code> field.

Table 44-20: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		0 Single Addr Comparator 0
		1 Single Addr Comparator 1
		2 Single Addr Comparator 2
		3 Single Addr Comparator 3
		4 Single Addr Comparator 4
		5 Single Addr Comparator 5
		6 Single Addr Comparator 6
		7 Single Addr Comparator 7
		8 Single Addr Comparator 8
		9 Single Addr Comparator 9
		10 Single Addr Comparator 10
		11 Single Addr Comparator 11
		12 Single Addr Comparator 12
		13 Single Addr Comparator 13
		14 Single Addr Comparator 14
		15 Single Addr Comparator 15
		16 Addr Range Comparator 0
		17 Addr Range Comparator 1
		18 Addr Range Comparator 2
		19 Addr Range Comparator 3
		20 Addr Range Comparator 4
		21 Addr Range Comparator 5
		22 Addr Range Comparator 6
		23 Addr Range Comparator 7
		64 Counter 0 at Zero
		65 Counter 1 at Zero
		66 Counter 2 at Zero
		67 Counter 3 at Zero
		88 Context ID Comparator 0
		89 Context ID Comparator 1

Table 44-20: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		90	Context ID Comparator 2
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Counter Reload Event Register

The `CSPFT_CNTRLDEVR[n]` register defines the event that causes the corresponding counter to be reloaded with the value held in the corresponding `CSPFT_CNTRLDVR[n]` register.

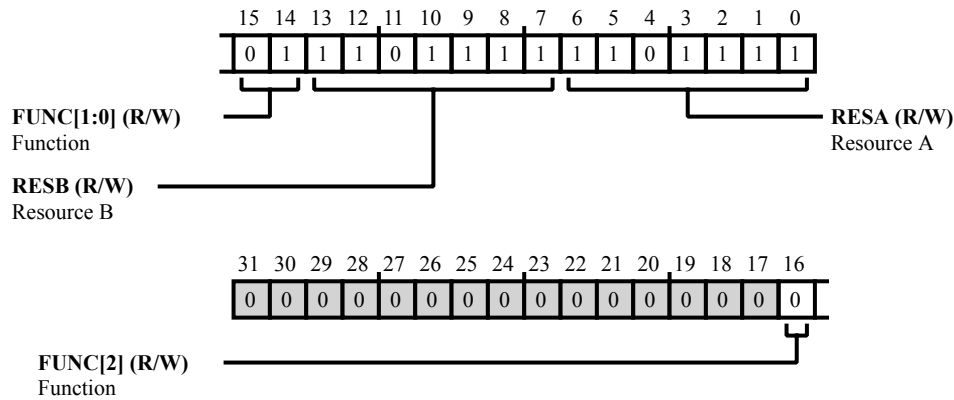


Figure 44-17: CSPFT_CNTRLDEVR[n] Register Diagram

Table 44-21: CSPFT_CNTRLDEVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:14 (R/W)	FUNC	Function. The <code>CSPFT_CNTRLDEVR[n].FUNC</code> bit field specifies the logical operation that combines the two resources that define the event.
		0 A
		1 NOT(A)
		2 A AND B
		3 NOT(A) AND B
		4 NOT(A) AND NOT(B)
		5 A OR B
		6 NOT(A) OR B
7 NOT(A) OR NOT(B)		
13:7 (R/W)	RESB	Resource B. The <code>CSPFT_CNTRLDEVR[n].RESB</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_CNTRLDEVR[n].FUNC</code> field (See <code>CSPFT_CNTRLDEVR[n].RESA</code> for list of possible values).

Table 44-21: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	RESA	Resource A. The CSPFT_CNTRLDEVR[n].RESA bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_CNTRLDEVR[n].FUNC field.
		0 Single Addr Comparator 0
		1 Single Addr Comparator 1
		2 Single Addr Comparator 2
		3 Single Addr Comparator 3
		4 Single Addr Comparator 4
		5 Single Addr Comparator 5
		6 Single Addr Comparator 6
		7 Single Addr Comparator 7
		8 Single Addr Comparator 8
		9 Single Addr Comparator 9
		10 Single Addr Comparator 10
		11 Single Addr Comparator 11
		12 Single Addr Comparator 12
		13 Single Addr Comparator 13
		14 Single Addr Comparator 14
		15 Single Addr Comparator 15
		16 Addr Range Comparator 0
		17 Addr Range Comparator 1
		18 Addr Range Comparator 2
		19 Addr Range Comparator 3
		20 Addr Range Comparator 4
		21 Addr Range Comparator 5
		22 Addr Range Comparator 6
23 Addr Range Comparator 7		
64 Counter 0 at zero		
65 Counter 1 at zero		
66 Counter 2 at zero		

Table 44-21: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		67	Counter 3 at zero
		88	Context ID comparator 0
		89	Context ID comparator 1
		90	Context ID comparator 2
		95	TraceEnable start/stop resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace prohibited
		111	Always TRUE

Counter Reload Value Register

The `CSPFT_CNTRLDVR[n]` register specifies the starting value of the corresponding counter.

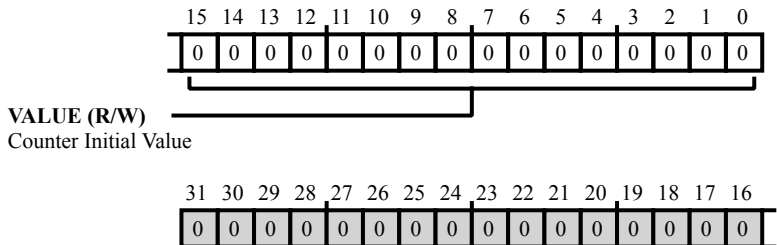


Figure 44-18: CSPFT_CNTRLDVR[n] Register Diagram

Table 44-22: CSPFT_CNTRLDVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Counter Initial Value. The <code>CSPFT_CNTRLDVR[n].VALUE</code> bit field specifies the starting value of the corresponding counter.

Counter Value Register

The `CSPFT_CNTVR[n]` register holds the current value of the corresponding counter.

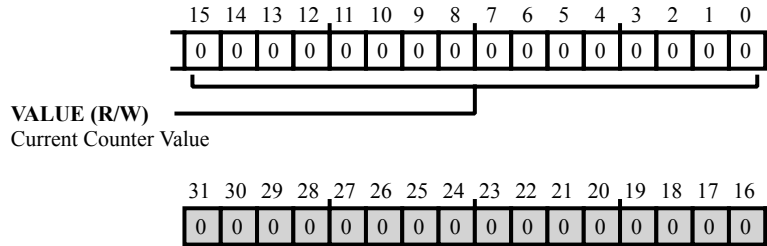


Figure 44-19: CSPFT_CNTVR[n] Register Diagram

Table 44-23: CSPFT_CNTVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Current Counter Value. The <code>CSPFT_CNTVR[n].VALUE</code> bit field specifies the current value of the corresponding counter.

Main Control Register

The `CSPFT_CTL` register controls general operation of the PTM, such as whether tracing is enabled.

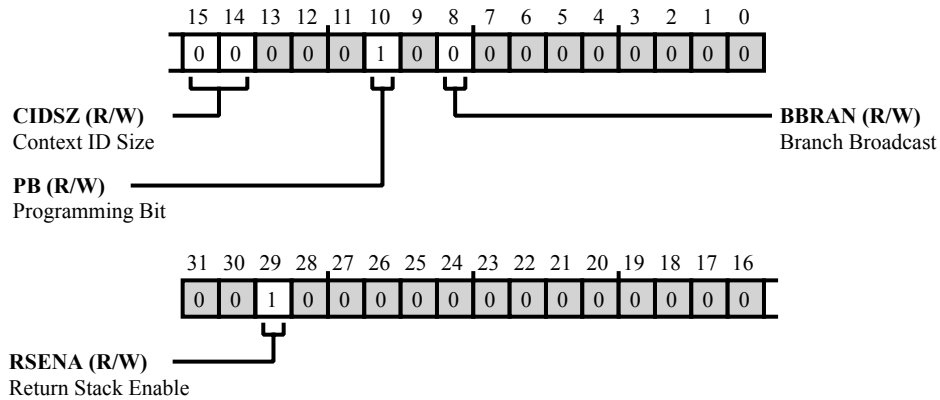


Figure 44-20: CSPFT_CTL Register Diagram

Table 44-24: CSPFT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	RSENA	Return Stack Enable. When the <code>CSPFT_CTL.RSENA</code> bit is set, the first indirect branch back to an address generates a branch without exception packet, and subsequent branches back to the same address generate E Atoms. This compresses inner loops of HW loops and code that indirectly branches back.
15:14 (R/W)	CIDSZ	Context ID Size. The <code>CSPFT_CTL.CIDSZ</code> bit field specifies the byte size to trace. Only the bytes specified are traced, even if the new Context ID value is larger than this.
		0 No Context ID Tracing
		1 One byte Traced
		2 Two Bytes Traced
		3 Three Bytes Traced

Table 44-24: CSPFT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration				
10 (R/W)	PB	<p>Programming Bit.</p> <p>To program the CSPFT, use the following procedure.</p> <ol style="list-style-type: none"> 1. Set the <code>CSPFT_CTL.PB</code> bit to disable all trace functionality. 2. Poll the <code>CSPFT_STAT.PB</code> bit waiting for it to be 1 (FIFO drained, trace halted). 3. Program the trace registers, counter and other registers, as required. 4. Set this bit to 0. 5. Poll the <code>CSPFT_STAT.PB</code> bit until it reads 0 (trace status reset, trace restarted). <p>When the <code>CSPFT_CTL.PB</code> bit is set, the FIFO is drained and no more trace is produced. All counters are held in their present state and the external outputs are forced low. After the FIFO is drained, the <code>CSPFT_STAT.PB</code> is set to reflect that the part is ready to program.</p> <p>When this bit is cleared, the trace status is cleared and trace is restarted.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">0</td> <td>Trace Enabled</td> </tr> <tr> <td style="text-align: center;">1</td> <td>Trace Disabled</td> </tr> </table>	0	Trace Enabled	1	Trace Disabled
0	Trace Enabled					
1	Trace Disabled					
8 (R/W)	BBRAN	<p>Branch Broadcast.</p> <p>Set the <code>CSPFT_CTL.BBRAN</code> bit to 1 to enable branch broadcasting. Branch broadcasting traces the address of direct branch instructions rather than producing E atoms.</p>				

Device Type Identifier Register

The `CSPFT_DEVTYPE` register is read-only. It provides a debugger with information about the component when the part number field is not recognized. The debugger can then report this information.

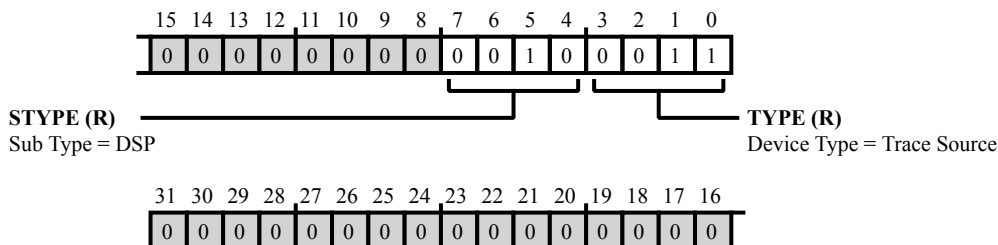


Figure 44-21: CSPFT_DEVTYPE Register Diagram

Table 44-25: CSPFT_DEVTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	STYPE	Sub Type = DSP.
3:0 (R/NW)	TYPE	Device Type = Trace Source.

External Output Event Register

The `CSPFT_EXTOUTEVR[n]` register defines the event that controls the corresponding `EXTOUT` external output signal.

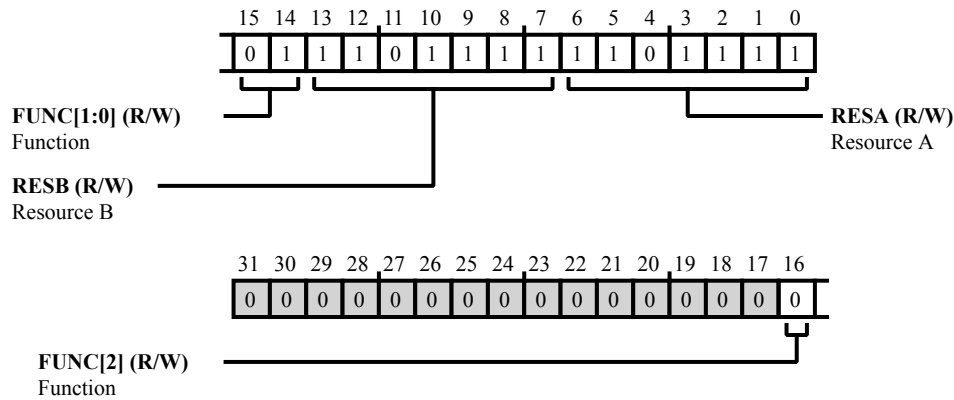


Figure 44-22: `CSPFT_EXTOUTEVR[n]` Register Diagram

Table 44-26: `CSPFT_EXTOUTEVR[n]` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:14 (R/W)	FUNC	Function. The <code>CSPFT_EXTOUTEVR[n].FUNC</code> bit field specifies the logical operation that combines the two resources that define the event.
		0 A
		1 NOT(A)
		2 A AND B
		3 NOT(A) AND B
		4 NOT(A) AND NOT(B)
		5 A OR B
		6 NOT(A) OR B
7 NOT(A) OR NOT(B)		
13:7 (R/W)	RESB	Resource B. The <code>CSPFT_EXTOUTEVR[n].RESB</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_EXTOUTEVR[n].FUNC</code> field (See <code>CSPFT_EXTOUTEVR[n].RESA</code> for list of possible values).

Table 44-26: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	RESA	Resource A. The CSPFT_EXTOUTEVR[n].RESA bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_EXTOUTEVR[n].FUNC field.
		0 Single Addr Comparator 0
		1 Single Addr Comparator 1
		2 Single Addr Comparator 2
		3 Single Addr Comparator 3
		4 Single Addr Comparator 4
		5 Single Addr Comparator 5
		6 Single Addr Comparator 6
		7 Single Addr Comparator 7
		8 Single Addr Comparator 8
		9 Single Addr Comparator 9
		10 Single Addr Comparator 10
		11 Single Addr Comparator 11
		12 Single Addr Comparator 12
		13 Single Addr Comparator 13
		14 Single Addr Comparator 14
		15 Single Addr Comparator 15
		16 Addr Range Comparator 0
		17 Addr Range Comparator 1
		18 Addr Range Comparator 2
		19 Addr Range Comparator 3
		20 Addr Range Comparator 4
		21 Addr Range Comparator 5
		22 Addr Range Comparator 6
23 Addr Range Comparator 7		
64 Counter 0 at Zero		
65 Counter 1 at Zero		
66 Counter 2 at Zero		

Table 44-26: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Hardware Feature Register

The `CSPFT_HWFEAT` register enables software to read the implementation defined configuration of the PTM, giving the number of each type of hardware resource. Each field indicates the number of instances of a particular resource, zero indicates that there are no implemented resources of that type.

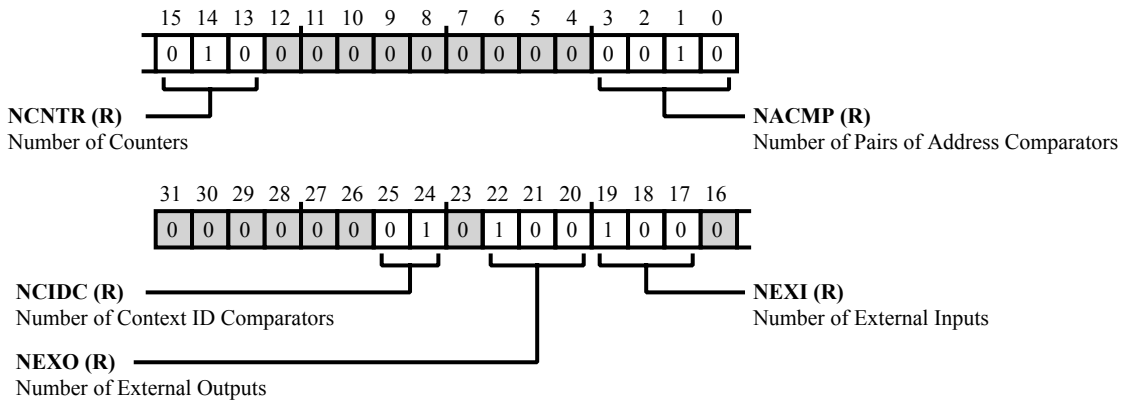


Figure 44-23: CSPFT_HWFEAT Register Diagram

Table 44-27: CSPFT_HWFEAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/NW)	NCIDC	Number of Context ID Comparators. The <code>CSPFT_HWFEAT.NCIDC</code> bit field identifies the number of context ID comparators.
22:20 (R/NW)	NEXO	Number of External Outputs. The <code>CSPFT_HWFEAT.NEXO</code> bit field identifies the number of external outputs (up to four).
19:17 (R/NW)	NEXI	Number of External Inputs. The <code>CSPFT_HWFEAT.NEXI</code> bit field identifies the number of external inputs (up to four).
15:13 (R/NW)	NCNTR	Number of Counters. The <code>CSPFT_HWFEAT.NCNTR</code> bit field identifies the number of counters (up to four) that are configured using the counter registers.
3:0 (R/NW)	NACMP	Number of Pairs of Address Comparators. The <code>CSPFT_HWFEAT.NACMP</code> bit field identifies the number of pairs of address comparators as address range comparators (ARCs). In this case, two adjacent address comparators form the ARC, so you can use address comparators 1 and 2 to define the first ARC. An ARC matches when any instruction in the specified range is committed for execution, regardless of whether the instruction passes its condition code test.

Table 44-27: CSPFT_HWFEAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	No Address Comparators
		1	1 Pair of Address Comparators
		2	2 Pairs of Address Comparators
		3	3 Pairs of Address Comparators
		4	4 Pairs of Address Comparators
		5	5 Pairs of Address Comparators
		6	6 Pairs of Address Comparators
		7	7 Pairs of Address Comparators
		8	8 Pairs of Address Comparators

Lock Access Register

The `CSPFT_LAR` register is used to provide lock and unlock access to all other CSPFT registers.

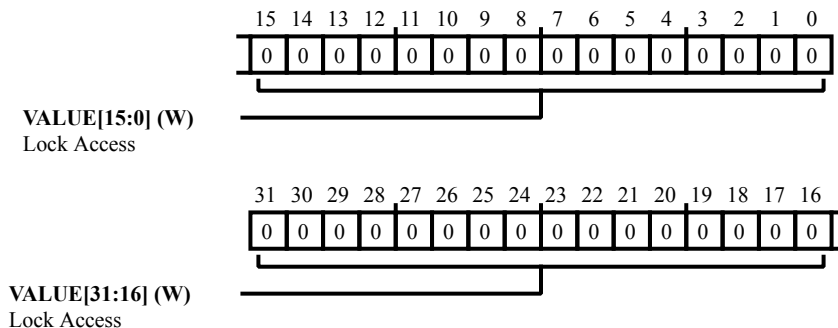


Figure 44-24: CSPFT_LAR Register Diagram

Table 44-28: CSPFT_LAR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Lock Access. Write 0xC5ACCE55 to the <code>CSPFT_LAR.VALUE</code> bit field to unlock. Write any other value to lock.

Lock Status Register

The `CSPFT_LSR` register is used to detect if the lock registers are implemented and if they are currently locked.

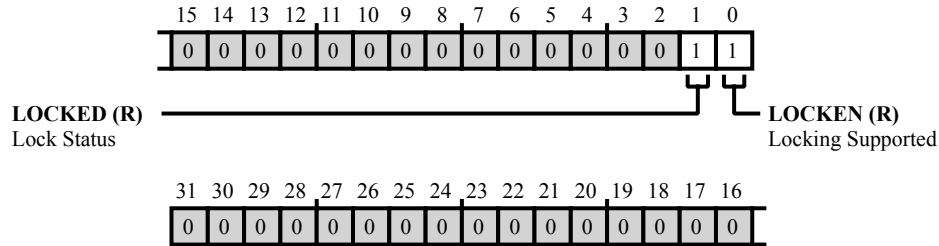


Figure 44-25: CSPFT_LSR Register Diagram

Table 44-29: CSPFT_LSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	LOCKED	Lock Status. The <code>CSPFT_LSR.LOCKED</code> bit indicates whether the PFT is locked.
		0 Writes are permitted
		1 Locked. Writes are ignored
0 (R/NW)	LOCKEN	Locking Supported. The <code>CSPFT_LSR.LOCKEN</code> bit indicates whether the lock registers are implemented for this interface.
		0 Locking is Not Required. This access is from an interface that ignores the lock registers.
		1 Locking is Required. This access is from an interface that requires the PFT to be unlocked.

Peripheral ID0 Register

The `CSPFT_PID0` register holds peripheral identification information.

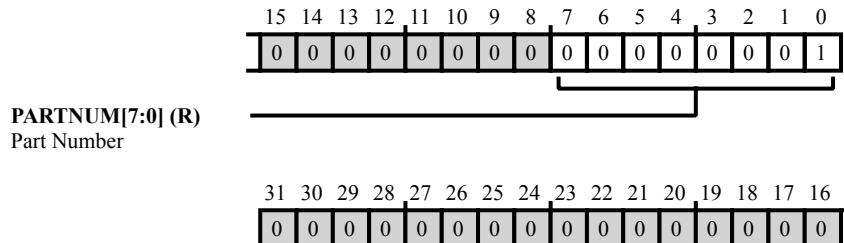


Figure 44-26: CSPFT_PID0 Register Diagram

Table 44-30: CSPFT_PID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PARTNUM	Part Number. The <code>CSPFT_PID0.PARTNUM</code> bit field holds the peripheral identification number.

Peripheral ID1 Register

The `CSPFT_PID1` register holds peripheral identification information.

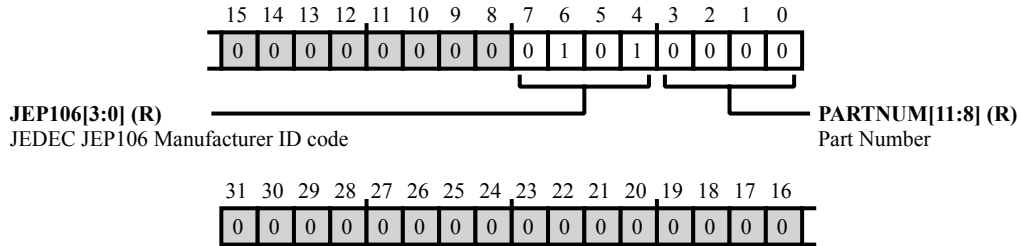


Figure 44-27: CSPFT_PID1 Register Diagram

Table 44-31: CSPFT_PID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.
3:0 (R/NW)	PARTNUM	Part Number. The <code>CSPFT_PID1</code> . <code>PARTNUM</code> bit field holds the peripheral identification number.

Peripheral ID2 Register

The `CSPFT_PID2` register holds peripheral identification information.

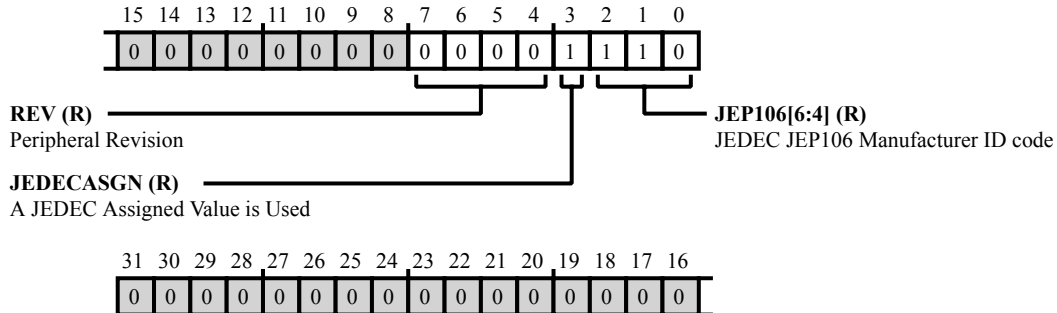


Figure 44-28: CSPFT_PID2 Register Diagram

Table 44-32: CSPFT_PID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV	Peripheral Revision.
3 (R/NW)	JEDECASGN	A JEDEC Assigned Value is Used. The <code>CSPFT_PID2</code> . <code>JEDECASGN</code> bit indicates that a JEDEC assigned value is used.
2:0 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.

Peripheral ID3 Register

The `CSPFT_PID3` register holds peripheral identification information.

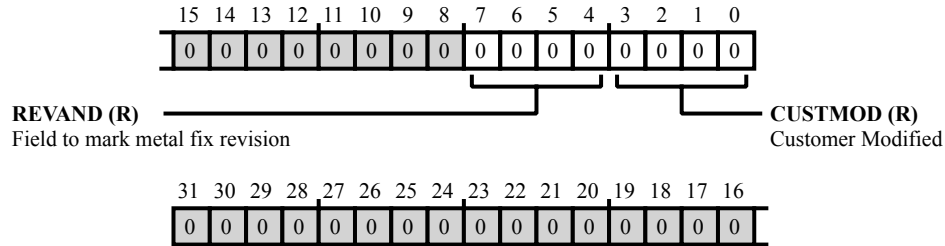


Figure 44-29: CSPFT_PID3 Register Diagram

Table 44-33: CSPFT_PID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REVAND	Field to mark metal fix revision.
3:0 (R/NW)	CUSTMOD	Customer Modified.

Peripheral ID4 Register

The `CSPFT_PID4` register holds peripheral identification information.

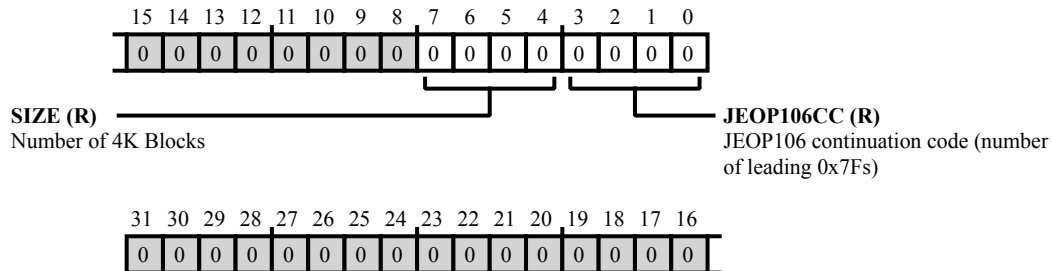


Figure 44-30: CSPFT_PID4 Register Diagram

Table 44-34: CSPFT_PID4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	SIZE	Number of 4K Blocks. The <code>CSPFT_PID4.SIZE</code> bit field contains the size of the component in 4K chunks minus 1 (for example 0=4K).
3:0 (R/NW)	JEOP106CC	JEOP106 continuation code (number of leading 0x7Fs).

Status Register

The `CSPFT_STAT` register provides information about the current status of the trace and trigger logic.

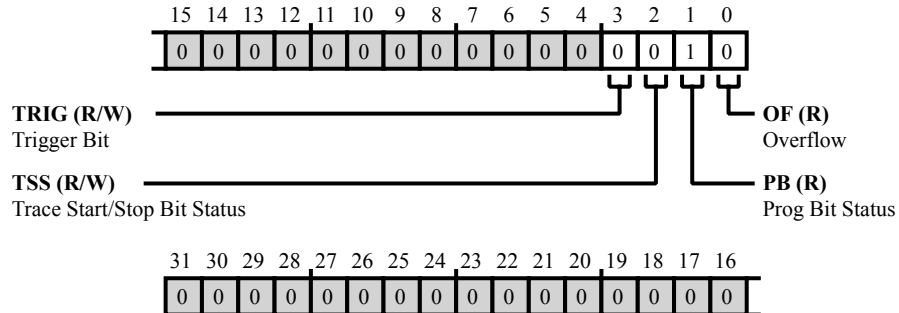


Figure 44-31: `CSPFT_STAT` Register Diagram

Table 44-35: `CSPFT_STAT` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TRIG	Trigger Bit. The <code>CSPFT_STAT.TRIG</code> bit is set when the trigger occurs, and prevents the trigger from being output until the CSPFT is next programmed. This bit is reset when the <code>CSPFT_CTL.PB</code> bit transitions from 1 to 0.
2 (R/W)	TSS	Trace Start/Stop Bit Status. The <code>CSPFT_STAT.TSS</code> bit holds the current status of the trace start/stop resource. If = 1, it indicates that a trace start address has been matched, without a corresponding trace stop address match. This bit =0 when trace is restarted (the <code>CSPFT_CTL.PB</code> bit transitions from 1 to 0).
1 (R/NW)	PB	Prog Bit Status. The <code>CSPFT_STAT.PB</code> bit indicates the current effective value of the <code>CSPFT_CTL.PB</code> bit. The program must wait for this bit to =1 before programming the CSPFT. (See the <code>CSPFT_CTL.PB</code> bit description).
0 (R/NW)	OF	Overflow. If the <code>CSPFT_STAT.OF</code> bit is =1, there is an overflow. This bit is cleared =0 when the trace is restarted (<code>CSPFT_CTL.PB</code> transitions from 1 to 0).

Synchronization Frequency Register

The `CSPFT_SYNCFR` register holds the trace synchronization frequency value.

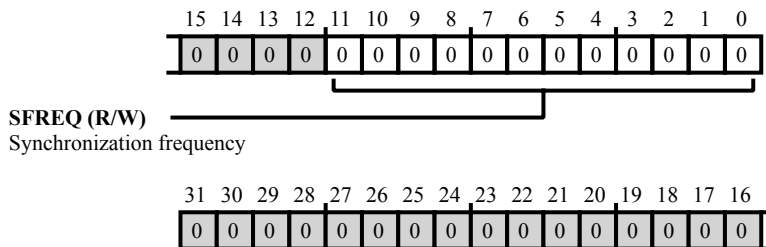


Figure 44-32: `CSPFT_SYNCFR` Register Diagram

Table 44-36: `CSPFT_SYNCFR` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	SFREQ	Synchronization frequency. The <code>CSPFT_SYNCFR.SFREQ</code> bit field is the number of 128-byte blocks of trace data after which you want to drop an address synchronization packet. If the circular buffer size is 16k, ensure that there are a few A-syncs in the buffer, so setting this to 16 means that every 2k there is an A-Sync packet.

TraceEnable Control Register

The `CSPFT_TECTL` register controls the start stop logic, whether resources specified are used for include or exclude, and specifies the address range comparators to use.

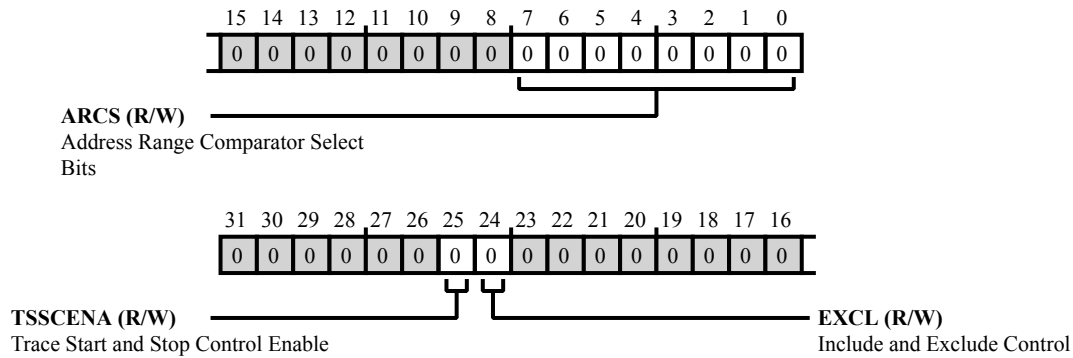


Figure 44-33: CSPFT_TECTL Register Diagram

Table 44-37: CSPFT_TECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	TSSCENA	Trace Start and Stop Control Enable.
		0 Tracing is not affected by the trace start/stop logic
		1 Tracing is controlled by the trace on and off address comparators
24 (R/W)	EXCL	Include and Exclude Control.
		0 Include. The specified address range comparators indicate the regions where tracing can occur. When outside the region trace is prevented.
		1 Exclude. The specified address range comparators indicate regions to be excluded from the trace. When outside the range tracing is enabled.
7:0 (R/W)	ARCS	Address Range Comparator Select Bits. When a bit in the <code>CSPFT_TECTL.ARCS</code> bit field is set to 1, it selects an address range comparator, for include/exclude control.

TraceEnable Event Register

The `CSPFT_TEEVENT` register defines the TraceEnable enabling event.

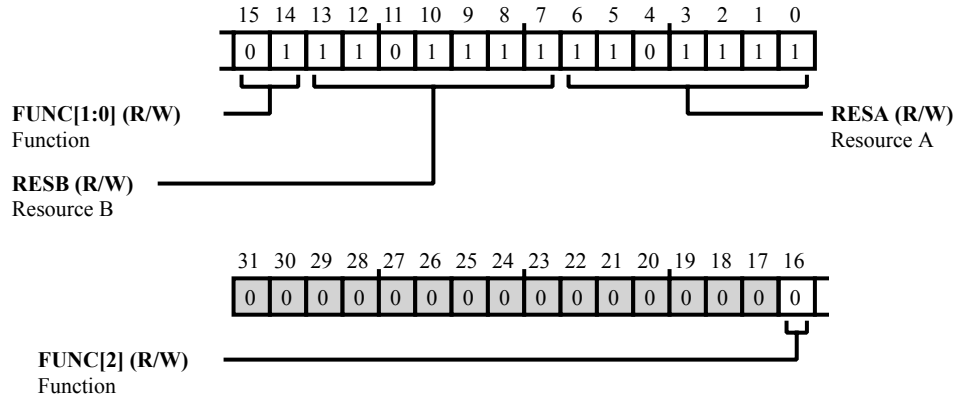


Figure 44-34: CSPFT_TEEVENT Register Diagram

Table 44-38: CSPFT_TEEVENT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:14 (R/W)	FUNC	Function. The <code>CSPFT_TEEVENT.FUNC</code> bit field specifies the logical operation that combines the two resources that define the event.
		0 A
		1 NOT(A)
		2 A AND B
		3 NOT(A) AND B
		4 NOT(A) AND NOT(B)
		5 A OR B
		6 NOT(A) OR B
7 NOT(A) OR NOT(B)		
13:7 (R/W)	RESB	Resource B. The <code>CSPFT_TEEVENT.RESB</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_TEEVENT.FUNC</code> field. (See <code>CSPFT_TEEVENT.RESA</code> for list of possible values).
6:0 (R/W)	RESA	Resource A. The <code>CSPFT_TEEVENT.RESA</code> bit field specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_TEEVENT.FUNC</code> field.
		0 Single Addr Comparator 0

Table 44-38: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		1 Single Addr Comparator 1
		2 Single Addr Comparator 2
		3 Single Addr Comparator 3
		4 Single Addr Comparator 4
		5 Single Addr Comparator 5
		6 Single Addr Comparator 6
		7 Single Addr Comparator 7
		8 Single Addr Comparator 8
		9 Single Addr Comparator 9
		10 Single Addr Comparator 10
		11 Single Addr Comparator 11
		12 Single Addr Comparator 12
		13 Single Addr Comparator 13
		14 Single Addr Comparator 14
		15 Single Addr Comparator 15
		16 Addr Range Comparator 0
		17 Addr Range Comparator 1
		18 Addr Range Comparator 2
		19 Addr Range Comparator 3
		20 Addr Range Comparator 4
		21 Addr Range Comparator 5
		22 Addr Range Comparator 6
		23 Addr Range Comparator 7
		64 Counter 0 at Zero
		65 Counter 1 at Zero
		66 Counter 2 at Zero
		67 Counter 3 at Zero
		88 Context ID Comparator 0
		89 Context ID Comparator 1
		90 Context ID Comparator 2

Table 44-38: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

CoreSight Trace ID Register

The `CSPFT_TRACEIDR` register defines the 7-bit trace ID, for output to the trace bus.

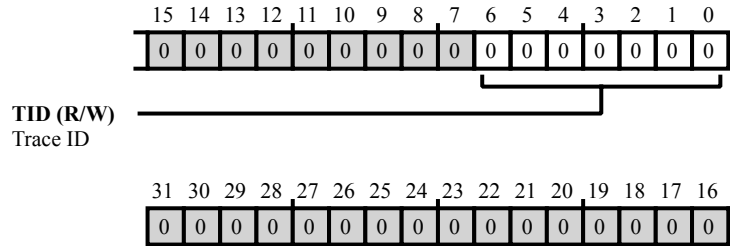


Figure 44-35: CSPFT_TRACEIDR Register Diagram

Table 44-39: CSPFT_TRACEIDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	TID	Trace ID. Defines the 7-bit trace ID, for output to the trace bus. Used in systems where multiple trace sources are present and tracing simultaneously. For example, when outputs trace onto the AMBA 3 Advanced Trace Bus, a unique ID is required for each trace source.

Trigger Event Register

The `CSPFT_TRIGGER` register defines the event that controls the trigger. This event creates the trigger output signal that is in the ATCLK domain.

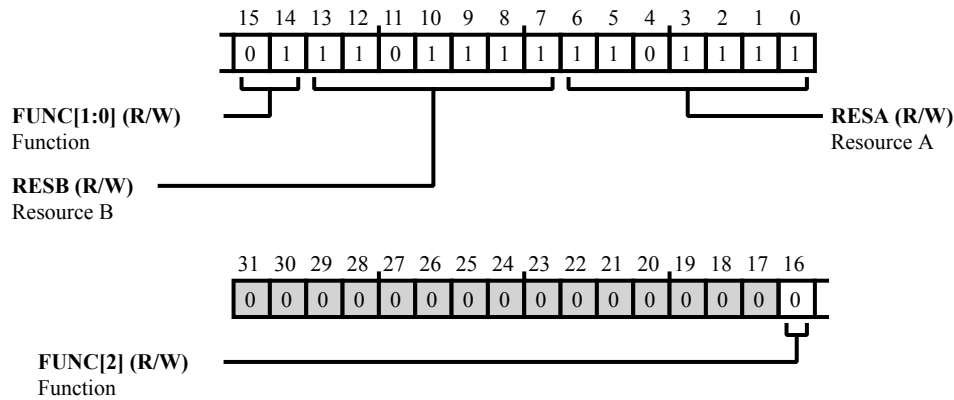


Figure 44-36: CSPFT_TRIGGER Register Diagram

Table 44-40: CSPFT_TRIGGER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:14 (R/W)	FUNC	Function. Specifies the logical operation that combines the two resources that define the event
		0 A
		1 NOT(A)
		2 A AND B
		3 NOT(A) AND B
		4 NOT(A) AND NOT(B)
		5 A OR B
		6 NOT(A) OR B
7 NOT(A) OR NOT(B)		
13:7 (R/W)	RESB	Resource B. Specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_TRIGGER.FUNC</code> field. (See <code>CSPFT_TRIGGER.RESA</code> for list of possible values.)
6:0 (R/W)	RESA	Resource A. Specifies one of the two resources that can be combined by the logical operation specified in the <code>CSPFT_TRIGGER.FUNC</code> field
		0 Single Addr Comparator 0

Table 44-40: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		1 Single Addr Comparator 1
		2 Single Addr Comparator 2
		3 Single Addr Comparator 3
		4 Single Addr Comparator 4
		5 Single Addr Comparator 5
		6 Single Addr Comparator 6
		7 Single Addr Comparator 7
		8 Single Addr Comparator 8
		9 Single Addr Comparator 9
		10 Single Addr Comparator 10
		11 Single Addr Comparator 11
		12 Single Addr Comparator 12
		13 Single Addr Comparator 13
		14 Single Addr Comparator 14
		15 Single Addr Comparator 15
		16 Addr Range Comparator 0
		17 Addr Range Comparator 1
		18 Addr Range Comparator 2
		19 Addr Range Comparator 3
		20 Addr Range Comparator 4
		21 Addr Range Comparator 5
		22 Addr Range Comparator 6
		23 Addr Range Comparator 7
		64 Counter 0 at Zero
		65 Counter 1 at Zero
		66 Counter 2 at Zero
		67 Counter 3 at Zero
		88 Context ID Comparator 0
		89 Context ID Comparator 1
		90 Context ID Comparator 2

Table 44-40: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

TraceEnable Start/Stop Control Register

The `CSPFT_TSSCTL` register specifies the single address comparators that hold the trace start and stop addresses.

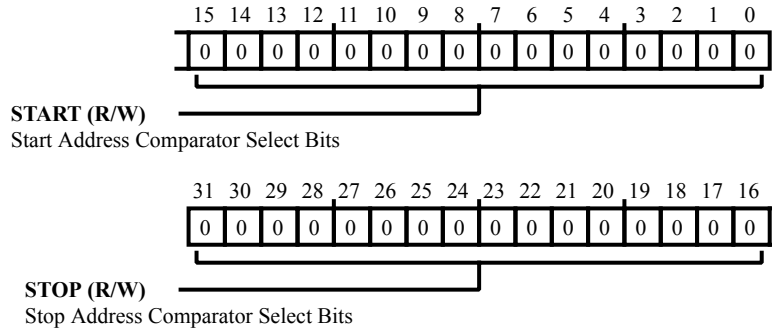


Figure 44-37: `CSPFT_TSSCTL` Register Diagram

Table 44-41: `CSPFT_TSSCTL` Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	STOP	Stop Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a stop address for the TraceEnable start/stop block.
		0 disabled
		1 Address Comparator 1
		2 Address Comparator 2
		4 Address Comparator 3
		8 Address Comparator 4
		16 Address Comparator 5
		32 Address Comparator 6
		64 Address Comparator 7
		128 Address Comparator 8
		256 Address Comparator 9
		512 Address Comparator 10
		1024 Address Comparator 11
		2048 Address Comparator 12
		4096 Address Comparator 13
		8192 Address Comparator 14
		16384 Address Comparator 15

Table 44-41: CSPFT_TSSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		32768 Address Comparator 16
15:0 (R/W)	START	Start Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a start address for the TraceEnable start/stop block.
		0 Disabled
		1 Address Comparator 1
		2 Address Comparator 2
		4 Address Comparator 3
		8 Address Comparator 4
		16 Address Comparator 5
		32 Address Comparator 6
		64 Address Comparator 7
		128 Address Comparator 8
		256 Address Comparator 9
		512 Address Comparator 10
		1024 Address Comparator 11
		2048 Address Comparator 12
		4096 Address Comparator 13
		8192 Address Comparator 14
		16384 Address Comparator 15
		32768 Address Comparator 16

ADSP-2156x TAPC Register Descriptions

TAPC (TAPC) contains the following registers.

Table 44-42: ADSP-2156x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control Register
TAPC_IDCODE	IDCODE Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register
TAPC_SDBGKEY1	Secure Debug Key 1 Register

Table 44-42: ADSP-2156x TAPC Register List (Continued)

Name	Description
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

Debug Control Register

The `TAPC_DBGCTL` register creates authentication signals to all CoreSight components in the system. This is a read write register accessible on the JTAG APB bus (not directly accessed by TAP).

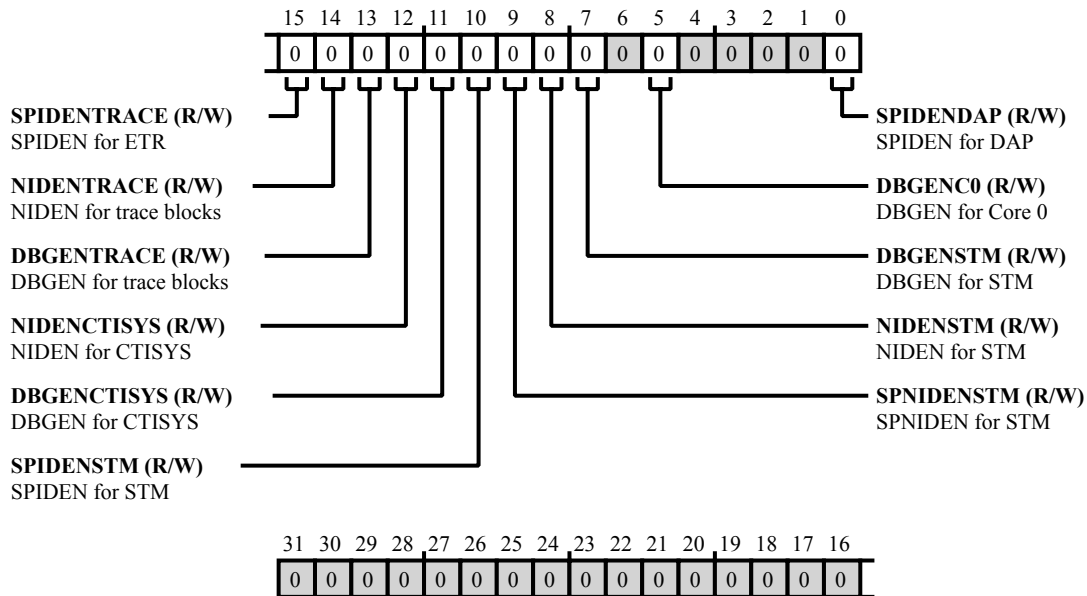


Figure 44-38: TAPC_DBGCTL Register Diagram

Table 44-43: TAPC_DBGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	SPIDENTRACE	SPIDEN for ETR. Secure Slave Enable Core0
14 (R/W)	NIDENTRACE	NIDEN for trace blocks. Non-Invasive Debug Enable for Trace Blocks
13 (R/W)	DBGENTRACE	DBGEN for trace blocks. DBGEN for Trace Blocks
12 (R/W)	NIDENCTISYS	NIDEN for CTISYS. NIDEN for System CTI
11 (R/W)	DBGENCTISYS	DBGEN for CTISYS. DBGEN for System CTI
10 (R/W)	SPIDENSTM	SPIDEN for STM. SPIDEN for STM
9 (R/W)	SPNIDENSTM	SPNIDEN for STM. SPINIDEN for STM

Table 44-43: TAPC_DBGCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	NIDENSTM	NIDEN for STM. NIDEN for STM
7 (R/W)	DBGENSTM	DBGEN for STM. DBGEN for STM
5 (R/W)	DBGENC0	DBGEN for Core 0. DBGEN for Core0
0 (R/W)	SPIDENDAP	SPIDEN for DAP. DBGEN for DAP

IDCODE Register

The `TAPC_IDCODE` register holds the IDCODE. The bit field is defined as follows.

$\text{IDCODE}[31:28] = 0x1^* - \text{REVID}$

$\text{IDCODE}[27:12] = 0x2820 - \text{JTAG ID}$

$\text{IDCODE}[11:1] = 0x65 - \text{Manufacturer ID}$

$\text{IDCODE}[0] = 0x1$

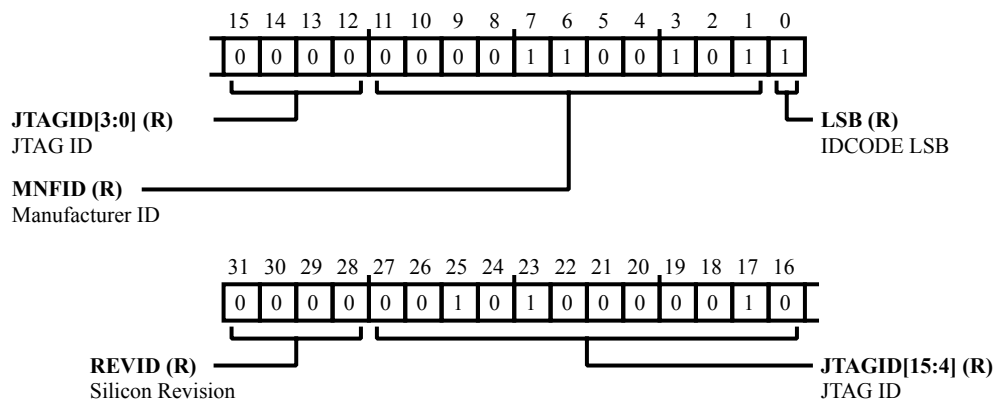


Figure 44-39: TAPC_IDCODE Register Diagram

Table 44-44: TAPC_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Silicon Revision. The <code>TAPC_IDCODE.REVID</code> bit field holds the silicon revision. See the processor anomaly list for details.
27:12 (R/NW)	JTAGID	JTAG ID.
11:1 (R/NW)	MNFID	Manufacturer ID.
0 (R/NW)	LSB	IDCODE LSB.

Secure Debug Key 0 Register

The `TAPC_SDBGKEY0` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. A debug key of 128 bits needs to be written into the Secure Debug Key registers (`TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, `TAPC_SDBGKEY3`) in the TAPC through the peripheral bus interface.

These registers hold the value of the key against which a matching key provided by the debug user is compared to enable a debug session. The task of writing these registers is performed (initially) by boot ROM code which copies a customer-selected key from the Flash memory info block to these registers.

An SDBGKEY value of all 0's is always an invalid key, a value of all 1's match the default value of the Secure Debug Key Compare registers and requires no entry in these registers. It is recommended programs have a significant number of 0's and 1's in a pseudo-random pattern throughout the 128-bit code for maximum protection.

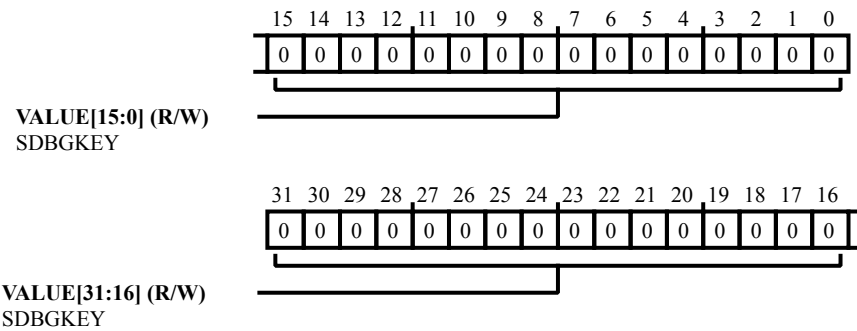


Figure 44-40: TAPC_SDBGKEY0 Register Diagram

Table 44-45: TAPC_SDBGKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The <code>TAPC_SDBGKEY0.VALUE</code> bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 1 Register

The `TAPC_SDBGKEY1` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the `TAPC_SDBGKEY0` register description for more information.

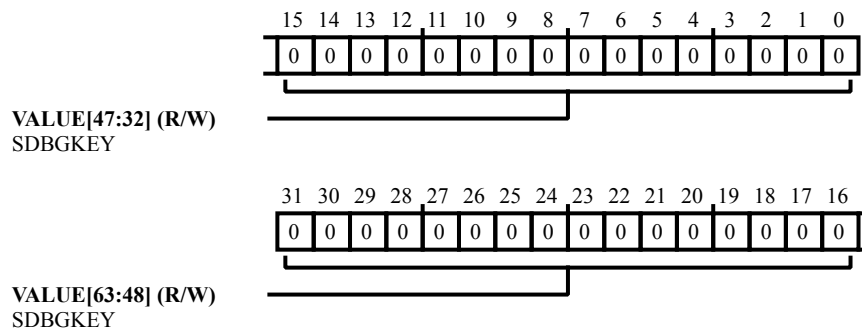


Figure 44-41: TAPC_SDBGKEY1 Register Diagram

Table 44-46: TAPC_SDBGKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The <code>TAPC_SDBGKEY1.VALUE</code> bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 2 Register

The `TAPC_SDBGKEY2` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the `TAPC_SDBGKEY0` register description for more information.

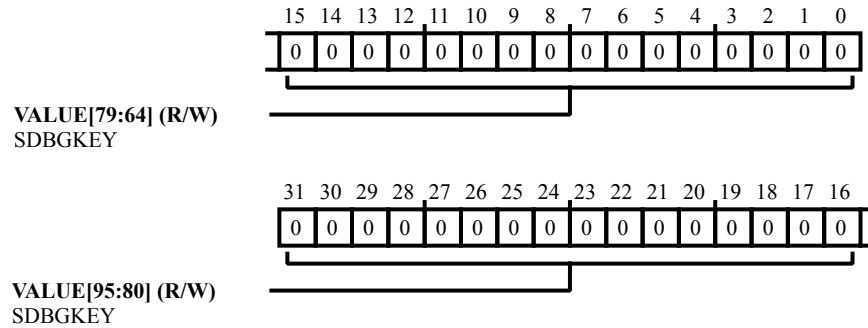


Figure 44-42: TAPC_SDBGKEY2 Register Diagram

Table 44-47: TAPC_SDBGKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The <code>TAPC_SDBGKEY2.VALUE</code> bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 3 Register

The `TAPC_SDBGKEY3` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the `TAPC_SDBGKEY0` register description for more information.

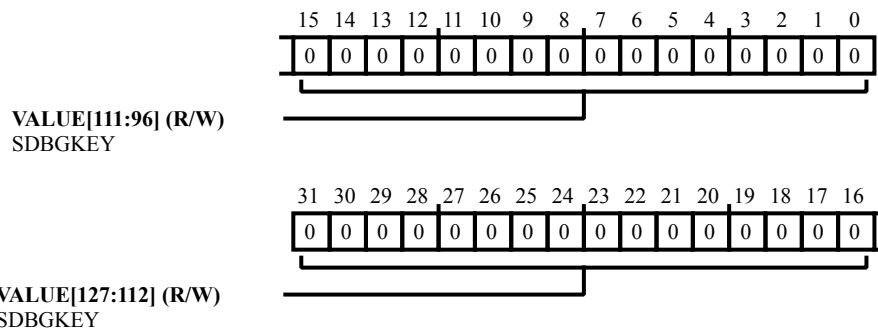


Figure 44-43: TAPC_SDBGKEY3 Register Diagram

Table 44-48: TAPC_SDBGKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The <code>TAPC_SDBGKEY3.VALUE</code> bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key Control Register

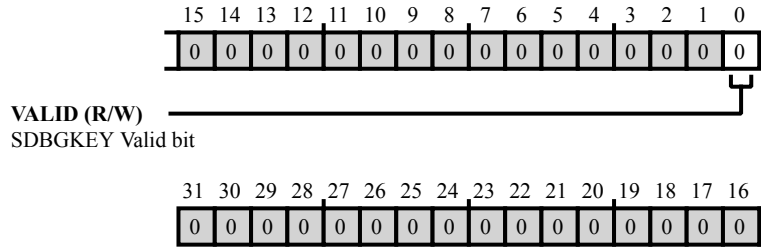


Figure 44-44: TAPC_SDBGKEY_CTL Register Diagram

Table 44-49: TAPC_SDBGKEY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALID	SDBGKEY Valid bit.

Secure Debug Key Status Register

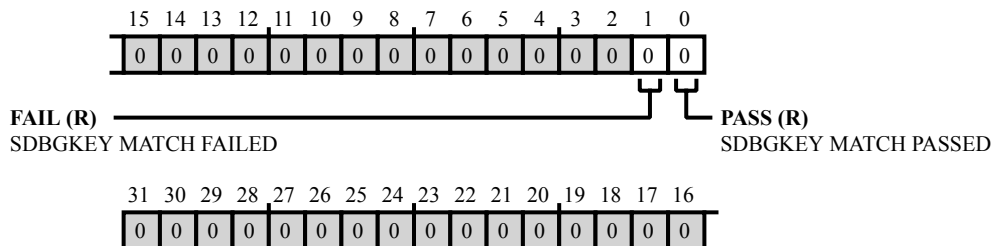


Figure 44-45: TAPC_SDBGKEY_STAT Register Diagram

Table 44-50: TAPC_SDBGKEY_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	FAIL	SDBGKEY MATCH FAILED.
0 (R/NW)	PASS	SDBGKEY MATCH PASSED.

USERCODE Register

The `TAPC_USERCODE` register

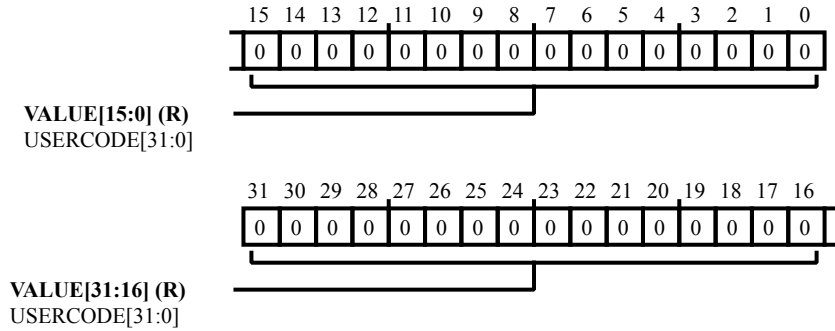


Figure 44-46: TAPC_USERCODE Register Diagram

Table 44-51: TAPC_USERCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	USERCODE[31:0].

Appendix A ADSP-2156x Register List

This appendix lists Memory-Mapped Register address and register names. The modules are presented in alphabetical order.

Table A-1: ADSP-2156x DMC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31071000	DMC0_DDR_LANE0_CTL0	DMC0 Data Lane 0 Control Register 0	0x00000000
0x31071004	DMC0_DDR_LANE0_CTL1	DMC0 Data Lane 0 Control Register 1	0x00000000
0x3107100C	DMC0_DDR_LANE1_CTL0	DMC0 Data Lane 1 Control Register 0	0x00000000
0x31071010	DMC0_DDR_LANE1_CTL1	DMC0 Data Lane 1 Control Register 1	0x00000000
0x31071018	DMC0_DDR_ROOT_CTL	DMC0 DDR ROOT Module Control Register	0x00000800
0x31071034	DMC0_DDR_ZQ_CTL0	DMC0 DDR Calibration Control Register 0	0xE0000000
0x31071038	DMC0_DDR_ZQ_CTL1	DMC0 DDR Calibration Control Register 1	0x00000000
0x3107103C	DMC0_DDR_ZQ_CTL2	DMC0 DDR Calibration Control Register 2	0x00000000
0x31071068	DMC0_DDR_CA_CTL	DMC0 DDR CA Lane Control Register	0x00000000

Table A-2: ADSP-2156x ASRC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C9240	ASRC0_CTL01	ASRC0 Control Register for ASRC 0 and 1	0x00000000
0x310C9244	ASRC0_CTL23	ASRC0 Control Register for ASRC 2 and 3	0x00000000
0x310C9248	ASRC0_MUTE	ASRC0 Mute Register	0x00000000
0x310C9260	ASRC0_RAT01	ASRC0 Ratio Register for ASRC 0 and 1	0x80008000
0x310C9264	ASRC0_RAT23	ASRC0 Ratio Register for ASRC 2 and 3	0x80008000

Table A-3: ADSP-2156x ASRC1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CA240	ASRC1_CTL01	ASRC1 Control Register for ASRC 0 and 1	0x00000000
0x310CA244	ASRC1_CTL23	ASRC1 Control Register for ASRC 2 and 3	0x00000000
0x310CA248	ASRC1_MUTE	ASRC1 Mute Register	0x00000000

Table A-3: ADSP-2156x ASRC1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CA260	ASRC1_RAT01	ASRC1 Ratio Register for ASRC 0 and 1	0x80008000
0x310CA264	ASRC1_RAT23	ASRC1 Ratio Register for ASRC 2 and 3	0x80008000

Table A-4: ADSP-2156x SCB1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30200020	SCB1_MST_IB_SYNC_MODE	SCB1 Master Synchronization Mode Register	0x00000004

Table A-5: ADSP-2156x CDU0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108F000	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F004	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F040	CDU0_STAT	CDU0 CDU Status	0x00000000
0x3108F044	CDU0_CLKINSEL	CDU0 CLKIN Select	0x00000000
0x3108F048	CDU0_REVID	CDU0 CDU Revision ID	0x00000011

Table A-6: ADSP-2156x CGU0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108D000	CGU0_CTL	CGU0 Control Register	0x00003C00
0x3108D004	CGU0_PLLCTL	CGU0 PLL Control Register	0x00000000
0x3108D008	CGU0_STAT	CGU0 Status Register	0x0000000F
0x3108D00C	CGU0_DIV	CGU0 Clocks Divisor Register	0x0A034482
0x3108D010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x3108D014	CGU0_OSCWDCTL	CGU0 Oscillator Watchdog Register	0x00007F00
0x3108D018	CGU0_TSCTL	CGU0 Time Stamp Control Register	0x00000000
0x3108D01C	CGU0_TSVALUE0	CGU0 Time Stamp Counter Initial 32 LSB Value Register	0x00000000
0x3108D020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x3108D024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x3108D028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000
0x3108D02C	CGU0_CCBF_DIS	CGU0 Core Clock Buffer Disable Register	0x00000000

Table A-6: ADSP-2156x CGU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108D030	CGU0_CCBF_STAT	CGU0 Core Clock Buffer Status Register	0x00000000
0x3108D038	CGU0_SCBF_DIS	CGU0 System Clock Buffer Disable Register	0x00000000
0x3108D03C	CGU0_SCBF_STAT	CGU0 System Clock Buffer Status Register	0x00000000
0x3108D040	CGU0_DIVEX	CGU0 DIV Register Extension	0x00200030
0x3108D048	CGU0_REVID	CGU0 Revision ID Register	0x00000030

Table A-7: ADSP-2156x CGU1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108E000	CGU1_CTL	CGU1 Control Register	0x00003C00
0x3108E004	CGU1_PLLCTL	CGU1 PLL Control Register	0x00000000
0x3108E008	CGU1_STAT	CGU1 Status Register	0x0000000F
0x3108E00C	CGU1_DIV	CGU1 Clocks Divisor Register	0x0A034482
0x3108E038	CGU1_SCBF_DIS	CGU1 System Clock Buffer Disable Register	0x00000000
0x3108E03C	CGU1_SCBF_STAT	CGU1 System Clock Buffer Status Register	0x00000000
0x3108E048	CGU1_REVID	CGU1 Revision ID Register	0x00000030

Table A-8: ADSP-2156x CNT0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3100B000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x3100B004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x3100B008	CNT0_STAT	CNT0 Status Register	0x00000000
0x3100B00C	CNT0_CMD	CNT0 Command Register	0x00000000
0x3100B010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x3100B014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x3100B018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x3100B01C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000

Table A-9: ADSP-2156x CRC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A5000	CRC0_CTL	CRC0 Control Register	0x00000000

Table A-9: ADSP-2156x CRC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A5004	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x310A5008	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x310A5014	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x310A5018	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000
0x310A501C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x310A5020	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000
0x310A5024	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x310A5028	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x310A502C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x310A5040	CRC0_STAT	CRC0 Status Register	0x00000000
0x310A5044	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x310A504C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x310A5050	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table A-10: ADSP-2156x CRC1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A6000	CRC1_CTL	CRC1 Control Register	0x00000000
0x310A6004	CRC1_DCNT	CRC1 Data Word Count Register	0x00000000
0x310A6008	CRC1_DCNTRLD	CRC1 Data Word Count Reload Register	0x00000000
0x310A6014	CRC1_COMP	CRC1 Data Compare Register	0x00000000
0x310A6018	CRC1_FILLVAL	CRC1 Fill Value Register	0x00000000
0x310A601C	CRC1_DFIFO	CRC1 Data FIFO Register	0x00000000
0x310A6020	CRC1_INEN	CRC1 Interrupt Enable Register	0x00000000
0x310A6024	CRC1_INEN_SET	CRC1 Interrupt Enable Set Register	0x00000000
0x310A6028	CRC1_INEN_CLR	CRC1 Interrupt Enable Clear Register	0x00000000
0x310A602C	CRC1_POLY	CRC1 Polynomial Register	0x00000000
0x310A6040	CRC1_STAT	CRC1 Status Register	0x00000000
0x310A6044	CRC1_DCNTCAP	CRC1 Data Count Capture Register	0x00000000
0x310A604C	CRC1_RESULT_FIN	CRC1 CRC Final Result Register	0x00000000
0x310A6050	CRC1_RESULT_CUR	CRC1 CRC Current Result Register	0x00000000

Table A-11: ADSP-2156x CTI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31102000	CTI0_CTICONTROL	CTI0 CTI Control Register	0x00000000
0x31102010	CTI0_CTIINTACK	CTI0 CTI Interrupt Acknowledge Register	0x00000000
0x31102014	CTI0_CTIAPPSET	CTI0 CTI Application Trigger Set Register	0x00000000
0x31102018	CTI0_CTIAPPCLEAR	CTI0 CTI Application Trigger Clear Register	0x00000000
0x3110201C	CTI0_CTIAPPPULSE	CTI0 CTI Application Pulse Register	0x00000000
0x31102020	CTI0_CTIINEN0	CTI0 CTI Trigger 0 to Channel Enable Register	0x00000000
0x31102024	CTI0_CTIINEN1	CTI0 CTI Trigger 1 to Channel Enable Register	0x00000000
0x31102028	CTI0_CTIINEN2	CTI0 CTI Trigger 2 to Channel Enable Register	0x00000000
0x3110202C	CTI0_CTIINEN3	CTI0 CTI Trigger 3 to Channel Enable Register	0x00000000
0x31102030	CTI0_CTIINEN4	CTI0 CTI Trigger 4 to Channel Enable Register	0x00000000
0x31102034	CTI0_CTIINEN5	CTI0 CTI Trigger 5 to Channel Enable Register	0x00000000
0x31102038	CTI0_CTIINEN6	CTI0 CTI Trigger 6 to Channel Enable Register	0x00000000
0x3110203C	CTI0_CTIINEN7	CTI0 CTI Trigger 7 to Channel Enable Register	0x00000000
0x311020A0	CTI0_CTIOUTEN0	CTI0 CTI Channel to Trigger 0 Enable Register	0x00000000
0x311020A4	CTI0_CTIOUTEN1	CTI0 CTI Channel to Trigger 1 Enable Register	0x00000000
0x311020A8	CTI0_CTIOUTEN2	CTI0 CTI Channel to Trigger 2 Enable Register	0x00000000
0x311020AC	CTI0_CTIOUTEN3	CTI0 CTI Channel to Trigger 3 Enable Register	0x00000000
0x311020B0	CTI0_CTIOUTEN4	CTI0 CTI Channel to Trigger 4 Enable Register	0x00000000
0x311020B4	CTI0_CTIOUTEN5	CTI0 CTI Channel to Trigger 5 Enable Register	0x00000000
0x311020B8	CTI0_CTIOUTEN6	CTI0 CTI Channel to Trigger 6 Enable Register	0x00000000
0x311020BC	CTI0_CTIOUTEN7	CTI0 CTI Channel to Trigger 7 Enable Register	0x00000000
0x31102130	CTI0_CTITRIGINSTATUS	CTI0 CTI Trigger In Status Register	0x00000000
0x31102134	CTI0_CTITRIGOUTSTATUS	CTI0 CTI Trigger Out Status Register	0x00000000
0x31102138	CTI0_CTICHINSTATUS	CTI0 CTI Channel In Status Register	0x00000000
0x3110213C	CTI0_CTICHOUTSTATUS	CTI0 CTI Channel Out Status Register	0x00000000
0x31102140	CTI0_CTIGATE	CTI0 Enable CTI Channel Gate Register	0x0000000F
0x31102144	CTI0_ASICCTL	CTI0 External Multiplexor Control Register	0x00000000
0x31102EDC	CTI0_ITCHINACK	CTI0 ITCHINACK	0x00000000
0x31102EE0	CTI0_ITTRIGINACK	CTI0 ITTRIGINACK	0x00000000
0x31102EE4	CTI0_ITCHOUT	CTI0 ITCHOUT	0x00000000

Table A-11: ADSP-2156x CTI0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31102EE8	CTI0_ITTRIGOUT	CTI0 ITTRIGOUT	0x00000000
0x31102EEC	CTI0_ITCHOUTACK	CTI0 ITCHOUTACK	0x00000000
0x31102EF0	CTI0_ITTRIGOUTACK	CTI0 ITTRIGOUTACK	0x00000000
0x31102EF4	CTI0_ITCHIN	CTI0 ITCHIN	0x00000000
0x31102EF8	CTI0_ITTRIGIN	CTI0 ITTRIGIN	0x00000000
0x31102F00	CTI0_ITCTRL	CTI0 Integration Mode Control Register	0x00000000
0x31102FA0	CTI0_CLAIMSET	CTI0 Claim Tag Set Register	0x0000000F
0x31102FA4	CTI0_CLAIMCLR	CTI0 Claim Tag Clear Register	0x00000000
0x31102FB0	CTI0_LAR	CTI0 Lock Access Register	0x00000000
0x31102FB4	CTI0_LSR	CTI0 Lock Status Register	0x00000003
0x31102FB8	CTI0_AUTHSTATUS	CTI0 Authentication Status	0x00000005
0x31102FC8	CTI0_DEVID	CTI0 Device ID	0x00040800
0x31102FCC	CTI0_DEVTYPE	CTI0 Device Type	0x00000014
0x31102FD0	CTI0_PERIPHID4	CTI0 Peripheral ID4	0x00000004
0x31102FD4	CTI0_PERIPHID5	CTI0 Peripheral ID5	0x00000000
0x31102FD8	CTI0_PERIPHID6	CTI0 Peripheral ID6	0x00000000
0x31102FDC	CTI0_PERIPHID7	CTI0 Peripheral ID7	0x00000000
0x31102FE0	CTI0_PERIPHID0	CTI0 Peripheral ID0	0x00000006
0x31102FE4	CTI0_PERIPHID1	CTI0 Peripheral ID1	0x000000B9
0x31102FE8	CTI0_PERIPHID2	CTI0 Peripheral ID2	0x0000003B
0x31102FEC	CTI0_PERIPHID3	CTI0 Peripheral ID3	0x00000000
0x31102FF0	CTI0_COMPID0	CTI0 Component ID0	0x0000000D
0x31102FF4	CTI0_COMPID1	CTI0 Component ID1	0x00000090
0x31102FF8	CTI0_COMPID2	CTI0 Component ID2	0x00000005
0x31102FFC	CTI0_COMPID3	CTI0 Component ID3	0x000000B1

Table A-12: ADSP-2156x CTI1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3110D000	CTI1_CTICONTROL	CTI1 CTI Control Register	0x00000000
0x3110D010	CTI1_CTIINTACK	CTI1 CTI Interrupt Acknowledge Register	0x00000000

Table A-12: ADSP-2156x CTI1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3110D014	CTI1_CTIAPPSET	CTI1 CTI Application Trigger Set Register	0x00000000
0x3110D018	CTI1_CTIAPPCLEAR	CTI1 CTI Application Trigger Clear Register	0x00000000
0x3110D01C	CTI1_CTIAPPULSE	CTI1 CTI Application Pulse Register	0x00000000
0x3110D020	CTI1_CTIINEN0	CTI1 CTI Trigger 0 to Channel Enable Register	0x00000000
0x3110D024	CTI1_CTIINEN1	CTI1 CTI Trigger 1 to Channel Enable Register	0x00000000
0x3110D028	CTI1_CTIINEN2	CTI1 CTI Trigger 2 to Channel Enable Register	0x00000000
0x3110D02C	CTI1_CTIINEN3	CTI1 CTI Trigger 3 to Channel Enable Register	0x00000000
0x3110D030	CTI1_CTIINEN4	CTI1 CTI Trigger 4 to Channel Enable Register	0x00000000
0x3110D034	CTI1_CTIINEN5	CTI1 CTI Trigger 5 to Channel Enable Register	0x00000000
0x3110D038	CTI1_CTIINEN6	CTI1 CTI Trigger 6 to Channel Enable Register	0x00000000
0x3110D03C	CTI1_CTIINEN7	CTI1 CTI Trigger 7 to Channel Enable Register	0x00000000
0x3110D0A0	CTI1_CTIOUTEN0	CTI1 CTI Channel to Trigger 0 Enable Register	0x00000000
0x3110D0A4	CTI1_CTIOUTEN1	CTI1 CTI Channel to Trigger 1 Enable Register	0x00000000
0x3110D0A8	CTI1_CTIOUTEN2	CTI1 CTI Channel to Trigger 2 Enable Register	0x00000000
0x3110D0AC	CTI1_CTIOUTEN3	CTI1 CTI Channel to Trigger 3 Enable Register	0x00000000
0x3110D0B0	CTI1_CTIOUTEN4	CTI1 CTI Channel to Trigger 4 Enable Register	0x00000000
0x3110D0B4	CTI1_CTIOUTEN5	CTI1 CTI Channel to Trigger 5 Enable Register	0x00000000
0x3110D0B8	CTI1_CTIOUTEN6	CTI1 CTI Channel to Trigger 6 Enable Register	0x00000000
0x3110D0BC	CTI1_CTIOUTEN7	CTI1 CTI Channel to Trigger 7 Enable Register	0x00000000
0x3110D130	CTI1_CTITRIGINSTATUS	CTI1 CTI Trigger In Status Register	0x00000000
0x3110D134	CTI1_CTITRIGOUTSTATUS	CTI1 CTI Trigger Out Status Register	0x00000000
0x3110D138	CTI1_CTICHINSTATUS	CTI1 CTI Channel In Status Register	0x00000000
0x3110D13C	CTI1_CTICHOUTSTATUS	CTI1 CTI Channel Out Status Register	0x00000000
0x3110D140	CTI1_CTIGATE	CTI1 Enable CTI Channel Gate Register	0x0000000F
0x3110D144	CTI1_ASICCTL	CTI1 External Multiplexor Control Register	0x00000000
0x3110DEDC	CTI1_ITCHINACK	CTI1 ITCHINACK	0x00000000
0x3110DEE0	CTI1_ITTRIGINACK	CTI1 ITTRIGINACK	0x00000000
0x3110DEE4	CTI1_ITCHOUT	CTI1 ITCHOUT	0x00000000
0x3110DEE8	CTI1_ITTRIGOUT	CTI1 ITTRIGOUT	0x00000000
0x3110DEEC	CTI1_ITCHOUTACK	CTI1 ITCHOUTACK	0x00000000

Table A-12: ADSP-2156x CTI1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3110DEF0	CTI1_ITTRIGOUTACK	CTI1 ITTRIGOUTACK	0x00000000
0x3110DEF4	CTI1_ITCHIN	CTI1 ITCHIN	0x00000000
0x3110DEF8	CTI1_ITTRIGIN	CTI1 ITTRIGIN	0x00000000
0x3110DF00	CTI1_ITCTRL	CTI1 Integration Mode Control Register	0x00000000
0x3110DFA0	CTI1_CLAIMSET	CTI1 Claim Tag Set Register	0x0000000F
0x3110DFA4	CTI1_CLAIMCLR	CTI1 Claim Tag Clear Register	0x00000000
0x3110DFB0	CTI1_LAR	CTI1 Lock Access Register	0x00000000
0x3110DFB4	CTI1_LSR	CTI1 Lock Status Register	0x00000003
0x3110DFB8	CTI1_AUTHSTATUS	CTI1 Authentication Status	0x00000005
0x3110DFC8	CTI1_DEVID	CTI1 Device ID	0x00040800
0x3110DFCC	CTI1_DEVTYPE	CTI1 Device Type	0x00000014
0x3110DFD0	CTI1_PERIPHID4	CTI1 Peripheral ID4	0x00000004
0x3110DFD4	CTI1_PERIPHID5	CTI1 Peripheral ID5	0x00000000
0x3110DFD8	CTI1_PERIPHID6	CTI1 Peripheral ID6	0x00000000
0x3110DFDC	CTI1_PERIPHID7	CTI1 Peripheral ID7	0x00000000
0x3110DFE0	CTI1_PERIPHID0	CTI1 Peripheral ID0	0x00000006
0x3110DFE4	CTI1_PERIPHID1	CTI1 Peripheral ID1	0x000000B9
0x3110DFE8	CTI1_PERIPHID2	CTI1 Peripheral ID2	0x0000003B
0x3110DFEC	CTI1_PERIPHID3	CTI1 Peripheral ID3	0x00000000
0x3110DFF0	CTI1_COMPID0	CTI1 Component ID0	0x0000000D
0x3110DFF4	CTI1_COMPID1	CTI1 Component ID1	0x00000090
0x3110DFF8	CTI1_COMPID2	CTI1 Component ID2	0x00000005
0x3110DFFC	CTI1_COMPID3	CTI1 Component ID3	0x000000B1

Table A-13: ADSP-2156x CTI2 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3110C000	CTI2_CTICONTROL	CTI2 CTI Control Register	0x00000000
0x3110C010	CTI2_CTIINTACK	CTI2 CTI Interrupt Acknowledge Register	0x00000000
0x3110C014	CTI2_CTIAPPSET	CTI2 CTI Application Trigger Set Register	0x00000000
0x3110C018	CTI2_CTIAPPCLEAR	CTI2 CTI Application Trigger Clear Register	0x00000000

Table A-13: ADSP-2156x CTI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3110C01C	CTI2_CTIAPPPULSE	CTI2 CTI Application Pulse Register	0x00000000
0x3110C020	CTI2_CTIINEN0	CTI2 CTI Trigger 0 to Channel Enable Register	0x00000000
0x3110C024	CTI2_CTIINEN1	CTI2 CTI Trigger 1 to Channel Enable Register	0x00000000
0x3110C028	CTI2_CTIINEN2	CTI2 CTI Trigger 2 to Channel Enable Register	0x00000000
0x3110C02C	CTI2_CTIINEN3	CTI2 CTI Trigger 3 to Channel Enable Register	0x00000000
0x3110C030	CTI2_CTIINEN4	CTI2 CTI Trigger 4 to Channel Enable Register	0x00000000
0x3110C034	CTI2_CTIINEN5	CTI2 CTI Trigger 5 to Channel Enable Register	0x00000000
0x3110C038	CTI2_CTIINEN6	CTI2 CTI Trigger 6 to Channel Enable Register	0x00000000
0x3110C03C	CTI2_CTIINEN7	CTI2 CTI Trigger 7 to Channel Enable Register	0x00000000
0x3110C0A0	CTI2_CTIOUTEN0	CTI2 CTI Channel to Trigger 0 Enable Register	0x00000000
0x3110C0A4	CTI2_CTIOUTEN1	CTI2 CTI Channel to Trigger 1 Enable Register	0x00000000
0x3110C0A8	CTI2_CTIOUTEN2	CTI2 CTI Channel to Trigger 2 Enable Register	0x00000000
0x3110C0AC	CTI2_CTIOUTEN3	CTI2 CTI Channel to Trigger 3 Enable Register	0x00000000
0x3110C0B0	CTI2_CTIOUTEN4	CTI2 CTI Channel to Trigger 4 Enable Register	0x00000000
0x3110C0B4	CTI2_CTIOUTEN5	CTI2 CTI Channel to Trigger 5 Enable Register	0x00000000
0x3110C0B8	CTI2_CTIOUTEN6	CTI2 CTI Channel to Trigger 6 Enable Register	0x00000000
0x3110C0BC	CTI2_CTIOUTEN7	CTI2 CTI Channel to Trigger 7 Enable Register	0x00000000
0x3110C130	CTI2_CTITRIGINSTATUS	CTI2 CTI Trigger In Status Register	0x00000000
0x3110C134	CTI2_CTITRIGOUTSTATUS	CTI2 CTI Trigger Out Status Register	0x00000000
0x3110C138	CTI2_CTICHINSTATUS	CTI2 CTI Channel In Status Register	0x00000000
0x3110C13C	CTI2_CTICHOUTSTATUS	CTI2 CTI Channel Out Status Register	0x00000000
0x3110C140	CTI2_CTIGATE	CTI2 Enable CTI Channel Gate Register	0x0000000F
0x3110C144	CTI2_ASICCTL	CTI2 External Multiplexor Control Register	0x00000000
0x3110CEDC	CTI2_ITCHINACK	CTI2 ITCHINACK	0x00000000
0x3110CEE0	CTI2_ITTRIGINACK	CTI2 ITTRIGINACK	0x00000000
0x3110CEE4	CTI2_ITCHOUT	CTI2 ITCHOUT	0x00000000
0x3110CEE8	CTI2_ITTRIGOUT	CTI2 ITTRIGOUT	0x00000000
0x3110CEEC	CTI2_ITCHOUTACK	CTI2 ITCHOUTACK	0x00000000
0x3110CEF0	CTI2_ITTRIGOUTACK	CTI2 ITTRIGOUTACK	0x00000000
0x3110CEF4	CTI2_ITCHIN	CTI2 ITCHIN	0x00000000

Table A-13: ADSP-2156x CTI2 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3110CEF8	CTI2_ITTRIGIN	CTI2 ITTRIGIN	0x00000000
0x3110CF00	CTI2_ITCTRL	CTI2 Integration Mode Control Register	0x00000000
0x3110CFA0	CTI2_CLAIMSET	CTI2 Claim Tag Set Register	0x0000000F
0x3110CFA4	CTI2_CLAIMCLR	CTI2 Claim Tag Clear Register	0x00000000
0x3110CFB0	CTI2_LAR	CTI2 Lock Access Register	0x00000000
0x3110CFB4	CTI2_LSR	CTI2 Lock Status Register	0x00000003
0x3110CFB8	CTI2_AUTHSTATUS	CTI2 Authentication Status	0x00000005
0x3110CFC8	CTI2_DEVID	CTI2 Device ID	0x00040800
0x3110CFCC	CTI2_DEVTYPE	CTI2 Device Type	0x00000014
0x3110CFD0	CTI2_PERIPHID4	CTI2 Peripheral ID4	0x00000004
0x3110CFD4	CTI2_PERIPHID5	CTI2 Peripheral ID5	0x00000000
0x3110CFD8	CTI2_PERIPHID6	CTI2 Peripheral ID6	0x00000000
0x3110CFDC	CTI2_PERIPHID7	CTI2 Peripheral ID7	0x00000000
0x3110CFE0	CTI2_PERIPHID0	CTI2 Peripheral ID0	0x00000006
0x3110CFE4	CTI2_PERIPHID1	CTI2 Peripheral ID1	0x000000B9
0x3110CFE8	CTI2_PERIPHID2	CTI2 Peripheral ID2	0x0000003B
0x3110CFEC	CTI2_PERIPHID3	CTI2 Peripheral ID3	0x00000000
0x3110CFF0	CTI2_COMPID0	CTI2 Component ID0	0x0000000D
0x3110CFF4	CTI2_COMPID1	CTI2 Component ID1	0x00000090
0x3110CFF8	CTI2_COMPID2	CTI2 Component ID2	0x00000005
0x3110CFFC	CTI2_COMPID3	CTI2 Component ID3	0x000000B1

Table A-14: ADSP-2156x CSPFT0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31103000	CSPFT0_CTL	CSPFT0 Main Control Register	0x20000400
0x31103004	CSPFT0_HWFEAT	CSPFT0 Hardware Feature Register	0x01484002
0x31103008	CSPFT0_TRIGGER	CSPFT0 Trigger Event Register	0x000077EF
0x31103010	CSPFT0_STAT	CSPFT0 Status Register	0x00000002
0x31103018	CSPFT0_TSSCTL	CSPFT0 TraceEnable Start/Stop Control Register	0x00000000
0x31103020	CSPFT0_TEEVENT	CSPFT0 TraceEnable Event Register	0x000077EF

Table A-14: ADSP-2156x CSPFT0 MMR Register Addresses (Continued)

Memory Map- ped Address	Register Name	Description	Reset Value
0x31103024	CSPFT0_TECTL	CSPFT0 TraceEnable Control Register	0x00000000
0x31103040	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103044	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103048	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x3110304C	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103080	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103084	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103088	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x3110308C	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103140	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x31103144	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x31103150	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x31103154	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x31103160	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x31103164	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x31103170	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x31103174	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x311031A0	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031A4	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031A8	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031AC	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031B0	CSPFT0_CIDCVR[n]	CSPFT0 Context ID Comparator Value	0x00000000
0x311031BC	CSPFT0_CIDCMR	CSPFT0 Context ID Comparator Mask Register	0x00000000
0x311031E0	CSPFT0_SYNCFR	CSPFT0 Synchronization Frequency Register	0x00000000
0x311031E8	CSPFT0_CCER	CSPFT0 Configuration Code Extension Register	0x00800000
0x31103200	CSPFT0_TRACEIDR	CSPFT0 CoreSight Trace ID Register	0x00000000
0x31103FA0	CSPFT0_CLAIMSET	CSPFT0 Claim Tag Set Register	0x0000000F
0x31103FA4	CSPFT0_CLAIMCLR	CSPFT0 Claim Tag Clear Register	0x00000000
0x31103FB0	CSPFT0_LAR	CSPFT0 Lock Access Register	0x00000000
0x31103FB4	CSPFT0_LSR	CSPFT0 Lock Status Register	0x00000003
0x31103FB8	CSPFT0_AUTHSTATUS	CSPFT0 Authentication Status Register	0x00000080

Table A-14: ADSP-2156x CSPFT0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31103FCC	CSPFT0_DEVTYPE	CSPFT0 Device Type Identifier Register	0x00000023
0x31103FD0	CSPFT0_PID4	CSPFT0 Peripheral ID4 Register	0x00000000
0x31103FE0	CSPFT0_PID0	CSPFT0 Peripheral ID0 Register	0x00000001
0x31103FE4	CSPFT0_PID1	CSPFT0 Peripheral ID1 Register	0x00000050
0x31103FE8	CSPFT0_PID2	CSPFT0 Peripheral ID2 Register	0x0000000E
0x31103FEC	CSPFT0_PID3	CSPFT0 Peripheral ID3 Register	0x00000000
0x31103FF0	CSPFT0_CID0	CSPFT0 Component ID0 Register	0x0000000D
0x31103FF4	CSPFT0_CID1	CSPFT0 Component ID1 Register	0x00000090
0x31103FF8	CSPFT0_CID2	CSPFT0 Component ID2 Register	0x00000005
0x31103FFC	CSPFT0_CID3	CSPFT0 Component ID3 Register	0x000000B1

Table A-15: ADSP-2156x DAI0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C90C0	DAI0_CLK0	DAI0 Clock Routing Control Register 0	0x252630C2
0x310C90C4	DAI0_CLK1	DAI0 Clock Routing Control Register 1	0x3DEF7BDE
0x310C90C8	DAI0_CLK2	DAI0 Clock Routing Control Register 2	0x00007BDE
0x310C90CC	DAI0_CLK3	DAI0 Clock Routing Control Register 3	0x3C000000
0x310C90D0	DAI0_CLK4	DAI0 Clock Routing Control Register 4	0x3DEF03DE
0x310C90D4	DAI0_CLK5	DAI0 Clock Routing Control Register 5	0x000003DE
0x310C9100	DAI0_DAT0	DAI0 Serial Data Routing Control Register 0	0x08144040
0x310C9104	DAI0_DAT1	DAI0 Serial Data Routing Control Register 1	0x0F38B289
0x310C9108	DAI0_DAT2	DAI0 Serial Data Routing Control Register 2	0x00000450
0x310C910C	DAI0_DAT3	DAI0 Serial Data Routing Control Register 3	0x00000000
0x310C9110	DAI0_DAT4	DAI0 Serial Data Routing Control Register 4	0x00000000
0x310C9114	DAI0_DAT5	DAI0 Serial Data Routing Control Register 5	0x00000000
0x310C9118	DAI0_DAT6	DAI0 Serial Data Routing Control Register 6	0x00FBEBFE
0x310C9140	DAI0_FS0	DAI0 Frame Sync Routing Control Register 0	0x2736B4E3
0x310C9144	DAI0_FS1	DAI0 Frame Sync Routing Control Register 1	0x3DEF7BDE
0x310C9148	DAI0_FS2	DAI0 Frame Sync Routing Control Register 2	0x00007BDE
0x310C9150	DAI0_FS4	DAI0 Frame Sync Routing Control Register 4	0x000003DE

Table A-15: ADSP-2156x DAI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9180	DAI0_PIN0	DAI0 Pin Buffer Assignment Register 0	0x04C80A94
0x310C9184	DAI0_PIN1	DAI0 Pin Buffer Assignment Register 1	0x04E84B96
0x310C9188	DAI0_PIN2	DAI0 Pin Buffer Assignment Register 2	0x03668C98
0x310C9190	DAI0_PIN4	DAI0 Pin Buffer Assignment Register 4	0x05694F9E
0x310C91C0	DAI0_MISC0	DAI0 Miscellaneous Control Register 0	0x3DEF7BDE
0x310C91C4	DAI0_MISC1	DAI0 Miscellaneous Control Register 1	0x3DEF7BDE
0x310C91E0	DAI0_PBEN0	DAI0 Pin Buffer Enable Register 0	0x0E2482CA
0x310C91E4	DAI0_PBEN1	DAI0 Pin Buffer Enable Register 1	0x1348D30F
0x310C91E8	DAI0_PBEN2	DAI0 Pin Buffer Enable Register 2	0x1A5545D6
0x310C91EC	DAI0_PBEN3	DAI0 Pin Buffer Enable Register 3	0x1D71F79B
0x310C9200	DAI0_IMSK_FE	DAI0 Falling-Edge Interrupt Mask Register	0x00000000
0x310C9204	DAI0_IMSK_RE	DAI0 Rising-Edge Interrupt Mask Register	0x00000000
0x310C9210	DAI0_IMSK_PRI	DAI0 Core Interrupt Priority Assignment Register	0x00000000
0x310C9220	DAI0_IRPTL_H	DAI0 High Priority Interrupt Latch Register	0x00000000
0x310C9224	DAI0_IRPTL_L	DAI0 Low Priority Interrupt Latch Register	0x00000000
0x310C9230	DAI0_IRPTL_HS	DAI0 Shadow High Priority Interrupt Latch Register	0x00000000
0x310C9234	DAI0_IRPTL_LS	DAI0 Shadow Low Priority Interrupt Latch Register	0x00000000
0x310C92E4	DAI0_PIN_STAT	DAI0 Pin Status Register	0x00000000
0x310C92E8	DAI0_GBL_SP_EN	DAI0 Global SPORT Enable Register	0x00000000
0x310C92EC	DAI0_GBL_INT_EN	DAI0 Global SPORT Interrupt Grouping Register	0x00000000

Table A-16: ADSP-2156x DAI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310CA0C0	DAI1_CLK0	DAI1 Clock Routing Control Register 0	0x252630C2
0x310CA0C4	DAI1_CLK1	DAI1 Clock Routing Control Register 1	0x3DEF7BDE
0x310CA0C8	DAI1_CLK2	DAI1 Clock Routing Control Register 2	0x00007BDE
0x310CA0CC	DAI1_CLK3	DAI1 Clock Routing Control Register 3	0x3C000000
0x310CA0D0	DAI1_CLK4	DAI1 Clock Routing Control Register 4	0x3DEF03DE
0x310CA0D4	DAI1_CLK5	DAI1 Clock Routing Control Register 5	0x000003DE
0x310CA100	DAI1_DAT0	DAI1 Serial Data Routing Control Register 0	0x08144040

Table A-16: ADSP-2156x DAI1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CA104	DAI1_DAT1	DAI1 Serial Data Routing Control Register 1	0x0F38B289
0x310CA108	DAI1_DAT2	DAI1 Serial Data Routing Control Register 2	0x00000450
0x310CA10C	DAI1_DAT3	DAI1 Serial Data Routing Control Register 3	0x00000000
0x310CA110	DAI1_DAT4	DAI1 Serial Data Routing Control Register 4	0x00000000
0x310CA114	DAI1_DAT5	DAI1 Serial Data Routing Control Register 5	0x00000000
0x310CA118	DAI1_DAT6	DAI1 Serial Data Routing Control Register 6	0x00FBEBFE
0x310CA140	DAI1_FS0	DAI1 Frame Sync Routing Control Register 0	0x2736B4E3
0x310CA144	DAI1_FS1	DAI1 Frame Sync Routing Control Register 1	0x3DEF7BDE
0x310CA148	DAI1_FS2	DAI1 Frame Sync Routing Control Register 2	0x00007BDE
0x310CA150	DAI1_FS4	DAI1 Frame Sync Routing Control Register 4	0x000003DE
0x310CA180	DAI1_PIN0	DAI1 Pin Buffer Assignment Register 0	0x04C80A94
0x310CA184	DAI1_PIN1	DAI1 Pin Buffer Assignment Register 1	0x04E84B96
0x310CA188	DAI1_PIN2	DAI1 Pin Buffer Assignment Register 2	0x03668C98
0x310CA190	DAI1_PIN4	DAI1 Pin Buffer Assignment Register 4	0x05694F9E
0x310CA1C0	DAI1_MISC0	DAI1 Miscellaneous Control Register 0	0x3DEF7BDE
0x310CA1C4	DAI1_MISC1	DAI1 Miscellaneous Control Register 1	0x3DEF7BDE
0x310CA1E0	DAI1_PBEN0	DAI1 Pin Buffer Enable Register 0	0x0E2482CA
0x310CA1E4	DAI1_PBEN1	DAI1 Pin Buffer Enable Register 1	0x1348D30F
0x310CA1E8	DAI1_PBEN2	DAI1 Pin Buffer Enable Register 2	0x1A5545D6
0x310CA1EC	DAI1_PBEN3	DAI1 Pin Buffer Enable Register 3	0x1D71F79B
0x310CA200	DAI1_IMSK_FE	DAI1 Falling-Edge Interrupt Mask Register	0x00000000
0x310CA204	DAI1_IMSK_RE	DAI1 Rising-Edge Interrupt Mask Register	0x00000000
0x310CA210	DAI1_IMSK_PRI	DAI1 Core Interrupt Priority Assignment Register	0x00000000
0x310CA220	DAI1_IRPTL_H	DAI1 High Priority Interrupt Latch Register	0x00000000
0x310CA224	DAI1_IRPTL_L	DAI1 Low Priority Interrupt Latch Register	0x00000000
0x310CA230	DAI1_IRPTL_HS	DAI1 Shadow High Priority Interrupt Latch Register	0x00000000
0x310CA234	DAI1_IRPTL_LS	DAI1 Shadow Low Priority Interrupt Latch Register	0x00000000
0x310CA2E4	DAI1_PIN_STAT	DAI1 Pin Status Register	0x00000000
0x310CA2E8	DAI1_GBL_SP_EN	DAI1 Global SPORT Enable Register	0x00000000
0x310CA2EC	DAI1_GBL_INT_EN	DAI1 Global SPORT Interrupt Grouping Register	0x00000000

Table A-17: ADSP-2156x DAPROM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31100000	DAPROM0_ROMENTRY00	DAPROM0 ROM Entry 00	0x00001003
0x31100004	DAPROM0_ROMENTRY01	DAPROM0 ROM Entry 01	0x00002003
0x31100008	DAPROM0_ROMENTRY02	DAPROM0 ROM Entry 02	0x00003003
0x3110000C	DAPROM0_ROMENTRY03	DAPROM0 ROM Entry 03	0x00004003
0x31100010	DAPROM0_ROMENTRY04	DAPROM0 ROM Entry 04	0x00005003
0x31100014	DAPROM0_ROMENTRY05	DAPROM0 ROM Entry 05	0x00006003
0x31100018	DAPROM0_ROMENTRY06	DAPROM0 ROM Entry 06	0x00007003
0x3110001C	DAPROM0_ROMENTRY07	DAPROM0 ROM Entry 07	0x00008003
0x31100020	DAPROM0_ROMENTRY08	DAPROM0 ROM Entry 08	0x00009003
0x31100024	DAPROM0_ROMENTRY09	DAPROM0 ROM Entry 09	0x0000A003
0x31100028	DAPROM0_ROMENTRY10	DAPROM0 ROM Entry 10	0x0000B003
0x3110002C	DAPROM0_ROMENTRY11	DAPROM0 ROM Entry 11	0x0000C003
0x31100030	DAPROM0_ROMENTRY12	DAPROM0 ROM Entry 12	0x0000D003
0x31100034	DAPROM0_ROMENTRY13	DAPROM0 ROM Entry 13	0x00010003

Table A-18: ADSP-2156x DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022000	DMA0_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x31022004	DMA0_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x31022008	DMA0_CFG	DMA0 Configuration Register	0x00000000
0x3102200C	DMA0_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x31022010	DMA0_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x31022014	DMA0_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022018	DMA0_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022024	DMA0_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x31022028	DMA0_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x3102202C	DMA0_ADDR_CUR	DMA0 Current Address Register	0x00000000
0x31022030	DMA0_STAT	DMA0 Status Register	0x00006000
0x31022034	DMA0_XCNT_CUR	DMA0 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022038	DMA0_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000

Table A-18: ADSP-2156x DMA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31022040	DMA0_BWLCNT	DMA0 Bandwidth Limit Count Register	0x00000000
0x31022044	DMA0_BWLCNT_CUR	DMA0 Bandwidth Limit Count Current Register	0x00000000
0x31022048	DMA0_BWMCNT	DMA0 Bandwidth Monitor Count Register	0x00000000
0x3102204C	DMA0_BWMCNT_CUR	DMA0 Bandwidth Monitor Count Current Register	0x00000000

Table A-19: ADSP-2156x DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022080	DMA1_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x31022084	DMA1_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x31022088	DMA1_CFG	DMA1 Configuration Register	0x00000000
0x3102208C	DMA1_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x31022090	DMA1_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x31022094	DMA1_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022098	DMA1_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x310220A4	DMA1_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x310220A8	DMA1_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000
0x310220AC	DMA1_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x310220B0	DMA1_STAT	DMA1 Status Register	0x00006000
0x310220B4	DMA1_XCNT_CUR	DMA1 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310220B8	DMA1_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000
0x310220C0	DMA1_BWLCNT	DMA1 Bandwidth Limit Count Register	0x00000000
0x310220C4	DMA1_BWLCNT_CUR	DMA1 Bandwidth Limit Count Current Register	0x00000000
0x310220C8	DMA1_BWMCNT	DMA1 Bandwidth Monitor Count Register	0x00000000
0x310220CC	DMA1_BWMCNT_CUR	DMA1 Bandwidth Monitor Count Current Register	0x00000000

Table A-20: ADSP-2156x DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023000	DMA10_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000
0x31023004	DMA10_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000

Table A-20: ADSP-2156x DMA10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31023008	DMA10_CFG	DMA10 Configuration Register	0x00000000
0x3102300C	DMA10_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x31023010	DMA10_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x31023014	DMA10_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023018	DMA10_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x31023024	DMA10_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x31023028	DMA10_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x3102302C	DMA10_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x31023030	DMA10_STAT	DMA10 Status Register	0x00006000
0x31023034	DMA10_XCNT_CUR	DMA10 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31023038	DMA10_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000
0x31023040	DMA10_BWLCNT	DMA10 Bandwidth Limit Count Register	0x00000000
0x31023044	DMA10_BWLCNT_CUR	DMA10 Bandwidth Limit Count Current Register	0x00000000
0x31023048	DMA10_BWMCNT	DMA10 Bandwidth Monitor Count Register	0x00000000
0x3102304C	DMA10_BWMCNT_CUR	DMA10 Bandwidth Monitor Count Current Register	0x00000000

Table A-21: ADSP-2156x DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023080	DMA11_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x31023084	DMA11_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x31023088	DMA11_CFG	DMA11 Configuration Register	0x00000000
0x3102308C	DMA11_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000
0x31023090	DMA11_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x31023094	DMA11_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023098	DMA11_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x310230A4	DMA11_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000
0x310230A8	DMA11_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000

Table A-21: ADSP-2156x DMA11 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310230AC	DMA11_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x310230B0	DMA11_STAT	DMA11 Status Register	0x00006000
0x310230B4	DMA11_XCNT_CUR	DMA11 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310230B8	DMA11_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000
0x310230C0	DMA11_BWLCNT	DMA11 Bandwidth Limit Count Register	0x00000000
0x310230C4	DMA11_BWLCNT_CUR	DMA11 Bandwidth Limit Count Current Register	0x00000000
0x310230C8	DMA11_BWMCNT	DMA11 Bandwidth Monitor Count Register	0x00000000
0x310230CC	DMA11_BWMCNT_CUR	DMA11 Bandwidth Monitor Count Current Register	0x00000000

Table A-22: ADSP-2156x DMA12 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31023100	DMA12_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000
0x31023104	DMA12_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x31023108	DMA12_CFG	DMA12 Configuration Register	0x00000000
0x3102310C	DMA12_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x31023110	DMA12_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x31023114	DMA12_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023118	DMA12_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x31023124	DMA12_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x31023128	DMA12_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x3102312C	DMA12_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x31023130	DMA12_STAT	DMA12 Status Register	0x00006000
0x31023134	DMA12_XCNT_CUR	DMA12 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31023138	DMA12_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000
0x31023140	DMA12_BWLCNT	DMA12 Bandwidth Limit Count Register	0x00000000
0x31023144	DMA12_BWLCNT_CUR	DMA12 Bandwidth Limit Count Current Register	0x00000000
0x31023148	DMA12_BWMCNT	DMA12 Bandwidth Monitor Count Register	0x00000000
0x3102314C	DMA12_BWMCNT_CUR	DMA12 Bandwidth Monitor Count Current Register	0x00000000

Table A-23: ADSP-2156x DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023180	DMA13_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x31023184	DMA13_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000
0x31023188	DMA13_CFG	DMA13 Configuration Register	0x00000000
0x3102318C	DMA13_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x31023190	DMA13_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x31023194	DMA13_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023198	DMA13_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x310231A4	DMA13_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000
0x310231A8	DMA13_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x310231AC	DMA13_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x310231B0	DMA13_STAT	DMA13 Status Register	0x00006000
0x310231B4	DMA13_XCNT_CUR	DMA13 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310231B8	DMA13_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000
0x310231C0	DMA13_BWLCNT	DMA13 Bandwidth Limit Count Register	0x00000000
0x310231C4	DMA13_BWLCNT_CUR	DMA13 Bandwidth Limit Count Current Register	0x00000000
0x310231C8	DMA13_BWMCNT	DMA13 Bandwidth Monitor Count Register	0x00000000
0x310231CC	DMA13_BWMCNT_CUR	DMA13 Bandwidth Monitor Count Current Register	0x00000000

Table A-24: ADSP-2156x DMA14 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023200	DMA14_DSCPTR_NXT	DMA14 Pointer to Next Initial Descriptor Register	0x00000000
0x31023204	DMA14_ADDRSTART	DMA14 Start Address of Current Buffer Register	0x00000000
0x31023208	DMA14_CFG	DMA14 Configuration Register	0x00000000
0x3102320C	DMA14_XCNT	DMA14 Inner Loop Count Start Value Register	0x00000000
0x31023210	DMA14_XMOD	DMA14 Inner Loop Address Increment Register	0x00000000
0x31023214	DMA14_YCNT	DMA14 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023218	DMA14_YMOD	DMA14 Outer Loop Address Increment (2D only) Register	0x00000000

Table A-24: ADSP-2156x DMA14 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31023224	DMA14_DSCPTR_CUR	DMA14 Current Descriptor Pointer Register	0x00000000
0x31023228	DMA14_DSCPTR_PRV	DMA14 Previous Initial Descriptor Pointer Register	0x00000000
0x3102322C	DMA14_ADDR_CUR	DMA14 Current Address Register	0x00000000
0x31023230	DMA14_STAT	DMA14 Status Register	0x00006000
0x31023234	DMA14_XCNT_CUR	DMA14 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31023238	DMA14_YCNT_CUR	DMA14 Current Row Count (2D only) Register	0x00000000
0x31023240	DMA14_BWLCNT	DMA14 Bandwidth Limit Count Register	0x00000000
0x31023244	DMA14_BWLCNT_CUR	DMA14 Bandwidth Limit Count Current Register	0x00000000
0x31023248	DMA14_BWMCNT	DMA14 Bandwidth Monitor Count Register	0x00000000
0x3102324C	DMA14_BWMCNT_CUR	DMA14 Bandwidth Monitor Count Current Register	0x00000000

Table A-25: ADSP-2156x DMA15 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31023280	DMA15_DSCPTR_NXT	DMA15 Pointer to Next Initial Descriptor Register	0x00000000
0x31023284	DMA15_ADDRSTART	DMA15 Start Address of Current Buffer Register	0x00000000
0x31023288	DMA15_CFG	DMA15 Configuration Register	0x00000000
0x3102328C	DMA15_XCNT	DMA15 Inner Loop Count Start Value Register	0x00000000
0x31023290	DMA15_XMOD	DMA15 Inner Loop Address Increment Register	0x00000000
0x31023294	DMA15_YCNT	DMA15 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023298	DMA15_YMOD	DMA15 Outer Loop Address Increment (2D only) Register	0x00000000
0x310232A4	DMA15_DSCPTR_CUR	DMA15 Current Descriptor Pointer Register	0x00000000
0x310232A8	DMA15_DSCPTR_PRV	DMA15 Previous Initial Descriptor Pointer Register	0x00000000
0x310232AC	DMA15_ADDR_CUR	DMA15 Current Address Register	0x00000000
0x310232B0	DMA15_STAT	DMA15 Status Register	0x00006000
0x310232B4	DMA15_XCNT_CUR	DMA15 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310232B8	DMA15_YCNT_CUR	DMA15 Current Row Count (2D only) Register	0x00000000
0x310232C0	DMA15_BWLCNT	DMA15 Bandwidth Limit Count Register	0x00000000
0x310232C4	DMA15_BWLCNT_CUR	DMA15 Bandwidth Limit Count Current Register	0x00000000

Table A-25: ADSP-2156x DMA15 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310232C8	DMA15_BWMCNT	DMA15 Bandwidth Monitor Count Register	0x00000000
0x310232CC	DMA15_BWMCNT_CUR	DMA15 Bandwidth Monitor Count Current Register	0x00000000

Table A-26: ADSP-2156x DMA16 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023300	DMA16_DSCPTR_NXT	DMA16 Pointer to Next Initial Descriptor Register	0x00000000
0x31023304	DMA16_ADDRSTART	DMA16 Start Address of Current Buffer Register	0x00000000
0x31023308	DMA16_CFG	DMA16 Configuration Register	0x00000000
0x3102330C	DMA16_XCNT	DMA16 Inner Loop Count Start Value Register	0x00000000
0x31023310	DMA16_XMOD	DMA16 Inner Loop Address Increment Register	0x00000000
0x31023314	DMA16_YCNT	DMA16 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023318	DMA16_YMOD	DMA16 Outer Loop Address Increment (2D only) Register	0x00000000
0x31023324	DMA16_DSCPTR_CUR	DMA16 Current Descriptor Pointer Register	0x00000000
0x31023328	DMA16_DSCPTR_PRV	DMA16 Previous Initial Descriptor Pointer Register	0x00000000
0x3102332C	DMA16_ADDR_CUR	DMA16 Current Address Register	0x00000000
0x31023330	DMA16_STAT	DMA16 Status Register	0x00006000
0x31023334	DMA16_XCNT_CUR	DMA16 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31023338	DMA16_YCNT_CUR	DMA16 Current Row Count (2D only) Register	0x00000000
0x31023340	DMA16_BWLCNT	DMA16 Bandwidth Limit Count Register	0x00000000
0x31023344	DMA16_BWLCNT_CUR	DMA16 Bandwidth Limit Count Current Register	0x00000000
0x31023348	DMA16_BWMCNT	DMA16 Bandwidth Monitor Count Register	0x00000000
0x3102334C	DMA16_BWMCNT_CUR	DMA16 Bandwidth Monitor Count Current Register	0x00000000

Table A-27: ADSP-2156x DMA17 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31023380	DMA17_DSCPTR_NXT	DMA17 Pointer to Next Initial Descriptor Register	0x00000000
0x31023384	DMA17_ADDRSTART	DMA17 Start Address of Current Buffer Register	0x00000000
0x31023388	DMA17_CFG	DMA17 Configuration Register	0x00000000

Table A-27: ADSP-2156x DMA17 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102338C	DMA17_XCNT	DMA17 Inner Loop Count Start Value Register	0x00000000
0x31023390	DMA17_XMOD	DMA17 Inner Loop Address Increment Register	0x00000000
0x31023394	DMA17_YCNT	DMA17 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31023398	DMA17_YMOD	DMA17 Outer Loop Address Increment (2D only) Register	0x00000000
0x310233A4	DMA17_DSCPTR_CUR	DMA17 Current Descriptor Pointer Register	0x00000000
0x310233A8	DMA17_DSCPTR_PRV	DMA17 Previous Initial Descriptor Pointer Register	0x00000000
0x310233AC	DMA17_ADDR_CUR	DMA17 Current Address Register	0x00000000
0x310233B0	DMA17_STAT	DMA17 Status Register	0x00006000
0x310233B4	DMA17_XCNT_CUR	DMA17 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310233B8	DMA17_YCNT_CUR	DMA17 Current Row Count (2D only) Register	0x00000000
0x310233C0	DMA17_BWLCNT	DMA17 Bandwidth Limit Count Register	0x00000000
0x310233C4	DMA17_BWLCNT_CUR	DMA17 Bandwidth Limit Count Current Register	0x00000000
0x310233C8	DMA17_BWMCNT	DMA17 Bandwidth Monitor Count Register	0x00000000
0x310233CC	DMA17_BWMCNT_CUR	DMA17 Bandwidth Monitor Count Current Register	0x00000000

Table A-28: ADSP-2156x DMA18 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A7100	DMA18_DSCPTR_NXT	DMA18 Pointer to Next Initial Descriptor Register	0x00000000
0x310A7104	DMA18_ADDRSTART	DMA18 Start Address of Current Buffer Register	0x00000000
0x310A7108	DMA18_CFG	DMA18 Configuration Register	0x00000000
0x310A710C	DMA18_XCNT	DMA18 Inner Loop Count Start Value Register	0x00000000
0x310A7110	DMA18_XMOD	DMA18 Inner Loop Address Increment Register	0x00000000
0x310A7114	DMA18_YCNT	DMA18 Outer Loop Count Start Value (2D only) Register	0x00000000
0x310A7118	DMA18_YMOD	DMA18 Outer Loop Address Increment (2D only) Register	0x00000000
0x310A7124	DMA18_DSCPTR_CUR	DMA18 Current Descriptor Pointer Register	0x00000000
0x310A7128	DMA18_DSCPTR_PRV	DMA18 Previous Initial Descriptor Pointer Register	0x00000000
0x310A712C	DMA18_ADDR_CUR	DMA18 Current Address Register	0x00000000

Table A-28: ADSP-2156x DMA18 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A7130	DMA18_STAT	DMA18 Status Register	0x00006000
0x310A7134	DMA18_XCNT_CUR	DMA18 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310A7138	DMA18_YCNT_CUR	DMA18 Current Row Count (2D only) Register	0x00000000
0x310A7140	DMA18_BWLCNT	DMA18 Bandwidth Limit Count Register	0x00000000
0x310A7144	DMA18_BWLCNT_CUR	DMA18 Bandwidth Limit Count Current Register	0x00000000
0x310A7148	DMA18_BWMCNT	DMA18 Bandwidth Monitor Count Register	0x00000000
0x310A714C	DMA18_BWMCNT_CUR	DMA18 Bandwidth Monitor Count Current Register	0x00000000

Table A-29: ADSP-2156x DMA19 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A7180	DMA19_DSCPTR_NXT	DMA19 Pointer to Next Initial Descriptor Register	0x00000000
0x310A7184	DMA19_ADDRSTART	DMA19 Start Address of Current Buffer Register	0x00000000
0x310A7188	DMA19_CFG	DMA19 Configuration Register	0x00000000
0x310A718C	DMA19_XCNT	DMA19 Inner Loop Count Start Value Register	0x00000000
0x310A7190	DMA19_XMOD	DMA19 Inner Loop Address Increment Register	0x00000000
0x310A7194	DMA19_YCNT	DMA19 Outer Loop Count Start Value (2D only) Register	0x00000000
0x310A7198	DMA19_YMOD	DMA19 Outer Loop Address Increment (2D only) Register	0x00000000
0x310A71A4	DMA19_DSCPTR_CUR	DMA19 Current Descriptor Pointer Register	0x00000000
0x310A71A8	DMA19_DSCPTR_PRV	DMA19 Previous Initial Descriptor Pointer Register	0x00000000
0x310A71AC	DMA19_ADDR_CUR	DMA19 Current Address Register	0x00000000
0x310A71B0	DMA19_STAT	DMA19 Status Register	0x00006000
0x310A71B4	DMA19_XCNT_CUR	DMA19 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310A71B8	DMA19_YCNT_CUR	DMA19 Current Row Count (2D only) Register	0x00000000
0x310A71C0	DMA19_BWLCNT	DMA19 Bandwidth Limit Count Register	0x00000000
0x310A71C4	DMA19_BWLCNT_CUR	DMA19 Bandwidth Limit Count Current Register	0x00000000
0x310A71C8	DMA19_BWMCNT	DMA19 Bandwidth Monitor Count Register	0x00000000
0x310A71CC	DMA19_BWMCNT_CUR	DMA19 Bandwidth Monitor Count Current Register	0x00000000

Table A-30: ADSP-2156x DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022100	DMA2_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x31022104	DMA2_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x31022108	DMA2_CFG	DMA2 Configuration Register	0x00000000
0x3102210C	DMA2_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x31022110	DMA2_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x31022114	DMA2_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022118	DMA2_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022124	DMA2_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x31022128	DMA2_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000
0x3102212C	DMA2_ADDR_CUR	DMA2 Current Address Register	0x00000000
0x31022130	DMA2_STAT	DMA2 Status Register	0x00006000
0x31022134	DMA2_XCNT_CUR	DMA2 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022138	DMA2_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000
0x31022140	DMA2_BWLCNT	DMA2 Bandwidth Limit Count Register	0x00000000
0x31022144	DMA2_BWLCNT_CUR	DMA2 Bandwidth Limit Count Current Register	0x00000000
0x31022148	DMA2_BWMCNT	DMA2 Bandwidth Monitor Count Register	0x00000000
0x3102214C	DMA2_BWMCNT_CUR	DMA2 Bandwidth Monitor Count Current Register	0x00000000

Table A-31: ADSP-2156x DMA20 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026080	DMA20_DSCPTR_NXT	DMA20 Pointer to Next Initial Descriptor Register	0x00000000
0x31026084	DMA20_ADDRSTART	DMA20 Start Address of Current Buffer Register	0x00000000
0x31026088	DMA20_CFG	DMA20 Configuration Register	0x00000000
0x3102608C	DMA20_XCNT	DMA20 Inner Loop Count Start Value Register	0x00000000
0x31026090	DMA20_XMOD	DMA20 Inner Loop Address Increment Register	0x00000000
0x31026094	DMA20_YCNT	DMA20 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026098	DMA20_YMOD	DMA20 Outer Loop Address Increment (2D only) Register	0x00000000
0x310260A4	DMA20_DSCPTR_CUR	DMA20 Current Descriptor Pointer Register	0x00000000

Table A-31: ADSP-2156x DMA20 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310260A8	DMA20_DSCPTR_PRV	DMA20 Previous Initial Descriptor Pointer Register	0x00000000
0x310260AC	DMA20_ADDR_CUR	DMA20 Current Address Register	0x00000000
0x310260B0	DMA20_STAT	DMA20 Status Register	0x00006000
0x310260B4	DMA20_XCNT_CUR	DMA20 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310260B8	DMA20_YCNT_CUR	DMA20 Current Row Count (2D only) Register	0x00000000
0x310260C0	DMA20_BWLCNT	DMA20 Bandwidth Limit Count Register	0x00000000
0x310260C4	DMA20_BWLCNT_CUR	DMA20 Bandwidth Limit Count Current Register	0x00000000
0x310260C8	DMA20_BWMCNT	DMA20 Bandwidth Monitor Count Register	0x00000000
0x310260CC	DMA20_BWMCNT_CUR	DMA20 Bandwidth Monitor Count Current Register	0x00000000

Table A-32: ADSP-2156x DMA21 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31026000	DMA21_DSCPTR_NXT	DMA21 Pointer to Next Initial Descriptor Register	0x00000000
0x31026004	DMA21_ADDRSTART	DMA21 Start Address of Current Buffer Register	0x00000000
0x31026008	DMA21_CFG	DMA21 Configuration Register	0x00000000
0x3102600C	DMA21_XCNT	DMA21 Inner Loop Count Start Value Register	0x00000000
0x31026010	DMA21_XMOD	DMA21 Inner Loop Address Increment Register	0x00000000
0x31026014	DMA21_YCNT	DMA21 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026018	DMA21_YMOD	DMA21 Outer Loop Address Increment (2D only) Register	0x00000000
0x31026024	DMA21_DSCPTR_CUR	DMA21 Current Descriptor Pointer Register	0x00000000
0x31026028	DMA21_DSCPTR_PRV	DMA21 Previous Initial Descriptor Pointer Register	0x00000000
0x3102602C	DMA21_ADDR_CUR	DMA21 Current Address Register	0x00000000
0x31026030	DMA21_STAT	DMA21 Status Register	0x00006000
0x31026034	DMA21_XCNT_CUR	DMA21 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026038	DMA21_YCNT_CUR	DMA21 Current Row Count (2D only) Register	0x00000000
0x31026040	DMA21_BWLCNT	DMA21 Bandwidth Limit Count Register	0x00000000
0x31026044	DMA21_BWLCNT_CUR	DMA21 Bandwidth Limit Count Current Register	0x00000000
0x31026048	DMA21_BWMCNT	DMA21 Bandwidth Monitor Count Register	0x00000000

Table A-32: ADSP-2156x DMA21 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102604C	DMA21_BWMCNT_CUR	DMA21 Bandwidth Monitor Count Current Register	0x00000000

Table A-33: ADSP-2156x DMA22 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D000	DMA22_DSCPTR_NXT	DMA22 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D004	DMA22_ADDRSTART	DMA22 Start Address of Current Buffer Register	0x00000000
0x3102D008	DMA22_CFG	DMA22 Configuration Register	0x00000000
0x3102D00C	DMA22_XCNT	DMA22 Inner Loop Count Start Value Register	0x00000000
0x3102D010	DMA22_XMOD	DMA22 Inner Loop Address Increment Register	0x00000000
0x3102D014	DMA22_YCNT	DMA22 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D018	DMA22_YMOD	DMA22 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102D024	DMA22_DSCPTR_CUR	DMA22 Current Descriptor Pointer Register	0x00000000
0x3102D028	DMA22_DSCPTR_PRV	DMA22 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D02C	DMA22_ADDR_CUR	DMA22 Current Address Register	0x00000000
0x3102D030	DMA22_STAT	DMA22 Status Register	0x00006000
0x3102D034	DMA22_XCNT_CUR	DMA22 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D038	DMA22_YCNT_CUR	DMA22 Current Row Count (2D only) Register	0x00000000
0x3102D040	DMA22_BWLCNT	DMA22 Bandwidth Limit Count Register	0x00000000
0x3102D044	DMA22_BWLCNT_CUR	DMA22 Bandwidth Limit Count Current Register	0x00000000
0x3102D048	DMA22_BWMCNT	DMA22 Bandwidth Monitor Count Register	0x00000000
0x3102D04C	DMA22_BWMCNT_CUR	DMA22 Bandwidth Monitor Count Current Register	0x00000000

Table A-34: ADSP-2156x DMA23 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D080	DMA23_DSCPTR_NXT	DMA23 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D084	DMA23_ADDRSTART	DMA23 Start Address of Current Buffer Register	0x00000000
0x3102D088	DMA23_CFG	DMA23 Configuration Register	0x00000000
0x3102D08C	DMA23_XCNT	DMA23 Inner Loop Count Start Value Register	0x00000000

Table A-34: ADSP-2156x DMA23 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D090	DMA23_XMOD	DMA23 Inner Loop Address Increment Register	0x00000000
0x3102D094	DMA23_YCNT	DMA23 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D098	DMA23_YMOD	DMA23 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102D0A4	DMA23_DSCPTR_CUR	DMA23 Current Descriptor Pointer Register	0x00000000
0x3102D0A8	DMA23_DSCPTR_PRV	DMA23 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D0AC	DMA23_ADDR_CUR	DMA23 Current Address Register	0x00000000
0x3102D0B0	DMA23_STAT	DMA23 Status Register	0x00006000
0x3102D0B4	DMA23_XCNT_CUR	DMA23 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D0B8	DMA23_YCNT_CUR	DMA23 Current Row Count (2D only) Register	0x00000000
0x3102D0C0	DMA23_BWLCNT	DMA23 Bandwidth Limit Count Register	0x00000000
0x3102D0C4	DMA23_BWLCNT_CUR	DMA23 Bandwidth Limit Count Current Register	0x00000000
0x3102D0C8	DMA23_BWMCNT	DMA23 Bandwidth Monitor Count Register	0x00000000
0x3102D0CC	DMA23_BWMCNT_CUR	DMA23 Bandwidth Monitor Count Current Register	0x00000000

Table A-35: ADSP-2156x DMA24 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D100	DMA24_DSCPTR_NXT	DMA24 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D104	DMA24_ADDRSTART	DMA24 Start Address of Current Buffer Register	0x00000000
0x3102D108	DMA24_CFG	DMA24 Configuration Register	0x00000000
0x3102D10C	DMA24_XCNT	DMA24 Inner Loop Count Start Value Register	0x00000000
0x3102D110	DMA24_XMOD	DMA24 Inner Loop Address Increment Register	0x00000000
0x3102D114	DMA24_YCNT	DMA24 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D118	DMA24_YMOD	DMA24 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102D124	DMA24_DSCPTR_CUR	DMA24 Current Descriptor Pointer Register	0x00000000
0x3102D128	DMA24_DSCPTR_PRV	DMA24 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D12C	DMA24_ADDR_CUR	DMA24 Current Address Register	0x00000000
0x3102D130	DMA24_STAT	DMA24 Status Register	0x00006000

Table A-35: ADSP-2156x DMA24 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3102D134	DMA24_XCNT_CUR	DMA24 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D138	DMA24_YCNT_CUR	DMA24 Current Row Count (2D only) Register	0x00000000
0x3102D140	DMA24_BWLCNT	DMA24 Bandwidth Limit Count Register	0x00000000
0x3102D144	DMA24_BWLCNT_CUR	DMA24 Bandwidth Limit Count Current Register	0x00000000
0x3102D148	DMA24_BWMCNT	DMA24 Bandwidth Monitor Count Register	0x00000000
0x3102D14C	DMA24_BWMCNT_CUR	DMA24 Bandwidth Monitor Count Current Register	0x00000000

Table A-36: ADSP-2156x DMA25 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3102D180	DMA25_DSCPTR_NXT	DMA25 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D184	DMA25_ADDRSTART	DMA25 Start Address of Current Buffer Register	0x00000000
0x3102D188	DMA25_CFG	DMA25 Configuration Register	0x00000000
0x3102D18C	DMA25_XCNT	DMA25 Inner Loop Count Start Value Register	0x00000000
0x3102D190	DMA25_XMOD	DMA25 Inner Loop Address Increment Register	0x00000000
0x3102D194	DMA25_YCNT	DMA25 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D198	DMA25_YMOD	DMA25 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102D1A4	DMA25_DSCPTR_CUR	DMA25 Current Descriptor Pointer Register	0x00000000
0x3102D1A8	DMA25_DSCPTR_PRV	DMA25 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D1AC	DMA25_ADDR_CUR	DMA25 Current Address Register	0x00000000
0x3102D1B0	DMA25_STAT	DMA25 Status Register	0x00006000
0x3102D1B4	DMA25_XCNT_CUR	DMA25 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D1B8	DMA25_YCNT_CUR	DMA25 Current Row Count (2D only) Register	0x00000000
0x3102D1C0	DMA25_BWLCNT	DMA25 Bandwidth Limit Count Register	0x00000000
0x3102D1C4	DMA25_BWLCNT_CUR	DMA25 Bandwidth Limit Count Current Register	0x00000000
0x3102D1C8	DMA25_BWMCNT	DMA25 Bandwidth Monitor Count Register	0x00000000
0x3102D1CC	DMA25_BWMCNT_CUR	DMA25 Bandwidth Monitor Count Current Register	0x00000000

Table A-37: ADSP-2156x DMA26 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D200	DMA26_DSCPTR_NXT	DMA26 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D204	DMA26_ADDRSTART	DMA26 Start Address of Current Buffer Register	0x00000000
0x3102D208	DMA26_CFG	DMA26 Configuration Register	0x00000000
0x3102D20C	DMA26_XCNT	DMA26 Inner Loop Count Start Value Register	0x00000000
0x3102D210	DMA26_XMOD	DMA26 Inner Loop Address Increment Register	0x00000000
0x3102D214	DMA26_YCNT	DMA26 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D218	DMA26_YMOD	DMA26 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102D224	DMA26_DSCPTR_CUR	DMA26 Current Descriptor Pointer Register	0x00000000
0x3102D228	DMA26_DSCPTR_PRV	DMA26 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D22C	DMA26_ADDR_CUR	DMA26 Current Address Register	0x00000000
0x3102D230	DMA26_STAT	DMA26 Status Register	0x00006000
0x3102D234	DMA26_XCNT_CUR	DMA26 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D238	DMA26_YCNT_CUR	DMA26 Current Row Count (2D only) Register	0x00000000
0x3102D240	DMA26_BWLCNT	DMA26 Bandwidth Limit Count Register	0x00000000
0x3102D244	DMA26_BWLCNT_CUR	DMA26 Bandwidth Limit Count Current Register	0x00000000
0x3102D248	DMA26_BWMCNT	DMA26 Bandwidth Monitor Count Register	0x00000000
0x3102D24C	DMA26_BWMCNT_CUR	DMA26 Bandwidth Monitor Count Current Register	0x00000000

Table A-38: ADSP-2156x DMA27 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102D280	DMA27_DSCPTR_NXT	DMA27 Pointer to Next Initial Descriptor Register	0x00000000
0x3102D284	DMA27_ADDRSTART	DMA27 Start Address of Current Buffer Register	0x00000000
0x3102D288	DMA27_CFG	DMA27 Configuration Register	0x00000000
0x3102D28C	DMA27_XCNT	DMA27 Inner Loop Count Start Value Register	0x00000000
0x3102D290	DMA27_XMOD	DMA27 Inner Loop Address Increment Register	0x00000000
0x3102D294	DMA27_YCNT	DMA27 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102D298	DMA27_YMOD	DMA27 Outer Loop Address Increment (2D only) Register	0x00000000

Table A-38: ADSP-2156x DMA27 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3102D2A4	DMA27_DSCPTR_CUR	DMA27 Current Descriptor Pointer Register	0x00000000
0x3102D2A8	DMA27_DSCPTR_PRV	DMA27 Previous Initial Descriptor Pointer Register	0x00000000
0x3102D2AC	DMA27_ADDR_CUR	DMA27 Current Address Register	0x00000000
0x3102D2B0	DMA27_STAT	DMA27 Status Register	0x00006000
0x3102D2B4	DMA27_XCNT_CUR	DMA27 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102D2B8	DMA27_YCNT_CUR	DMA27 Current Row Count (2D only) Register	0x00000000
0x3102D2C0	DMA27_BWLCNT	DMA27 Bandwidth Limit Count Register	0x00000000
0x3102D2C4	DMA27_BWLCNT_CUR	DMA27 Bandwidth Limit Count Current Register	0x00000000
0x3102D2C8	DMA27_BWMCNT	DMA27 Bandwidth Monitor Count Register	0x00000000
0x3102D2CC	DMA27_BWMCNT_CUR	DMA27 Bandwidth Monitor Count Current Register	0x00000000

Table A-39: ADSP-2156x DMA3 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31022180	DMA3_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000
0x31022184	DMA3_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x31022188	DMA3_CFG	DMA3 Configuration Register	0x00000000
0x3102218C	DMA3_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x31022190	DMA3_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x31022194	DMA3_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022198	DMA3_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x310221A4	DMA3_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000
0x310221A8	DMA3_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x310221AC	DMA3_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x310221B0	DMA3_STAT	DMA3 Status Register	0x00006000
0x310221B4	DMA3_XCNT_CUR	DMA3 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310221B8	DMA3_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000
0x310221C0	DMA3_BWLCNT	DMA3 Bandwidth Limit Count Register	0x00000000
0x310221C4	DMA3_BWLCNT_CUR	DMA3 Bandwidth Limit Count Current Register	0x00000000
0x310221C8	DMA3_BWMCNT	DMA3 Bandwidth Monitor Count Register	0x00000000

Table A-39: ADSP-2156x DMA3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310221CC	DMA3_BWMCNT_CUR	DMA3 Bandwidth Monitor Count Current Register	0x00000000

Table A-40: ADSP-2156x DMA30 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFF000	DMA30_DSCPTR_NXT	DMA30 Pointer to Next Initial Descriptor Register	0x00000000
0x30FFF004	DMA30_ADDRSTART	DMA30 Start Address of Current Buffer Register	0x00000000
0x30FFF008	DMA30_CFG	DMA30 Configuration Register	0x00000000
0x30FFF00C	DMA30_XCNT	DMA30 Inner Loop Count Start Value Register	0x00000000
0x30FFF010	DMA30_XMOD	DMA30 Inner Loop Address Increment Register	0x00000000
0x30FFF014	DMA30_YCNT	DMA30 Outer Loop Count Start Value (2D only) Register	0x00000000
0x30FFF018	DMA30_YMOD	DMA30 Outer Loop Address Increment (2D only) Register	0x00000000
0x30FFF024	DMA30_DSCPTR_CUR	DMA30 Current Descriptor Pointer Register	0x00000000
0x30FFF028	DMA30_DSCPTR_PRV	DMA30 Previous Initial Descriptor Pointer Register	0x00000000
0x30FFF02C	DMA30_ADDR_CUR	DMA30 Current Address Register	0x00000000
0x30FFF030	DMA30_STAT	DMA30 Status Register	0x00006000
0x30FFF034	DMA30_XCNT_CUR	DMA30 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x30FFF038	DMA30_YCNT_CUR	DMA30 Current Row Count (2D only) Register	0x00000000
0x30FFF040	DMA30_BWLCNT	DMA30 Bandwidth Limit Count Register	0x00000000
0x30FFF044	DMA30_BWLCNT_CUR	DMA30 Bandwidth Limit Count Current Register	0x00000000
0x30FFF048	DMA30_BWMCNT	DMA30 Bandwidth Monitor Count Register	0x00000000
0x30FFF04C	DMA30_BWMCNT_CUR	DMA30 Bandwidth Monitor Count Current Register	0x00000000

Table A-41: ADSP-2156x DMA34 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026180	DMA34_DSCPTR_NXT	DMA34 Pointer to Next Initial Descriptor Register	0x00000000
0x31026184	DMA34_ADDRSTART	DMA34 Start Address of Current Buffer Register	0x00000000
0x31026188	DMA34_CFG	DMA34 Configuration Register	0x00000000
0x3102618C	DMA34_XCNT	DMA34 Inner Loop Count Start Value Register	0x00000000

Table A-41: ADSP-2156x DMA34 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026190	DMA34_XMOD	DMA34 Inner Loop Address Increment Register	0x00000000
0x31026194	DMA34_YCNT	DMA34 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026198	DMA34_YMOD	DMA34 Outer Loop Address Increment (2D only) Register	0x00000000
0x310261A4	DMA34_DSCPTR_CUR	DMA34 Current Descriptor Pointer Register	0x00000000
0x310261A8	DMA34_DSCPTR_PRV	DMA34 Previous Initial Descriptor Pointer Register	0x00000000
0x310261AC	DMA34_ADDR_CUR	DMA34 Current Address Register	0x00000000
0x310261B0	DMA34_STAT	DMA34 Status Register	0x00006000
0x310261B4	DMA34_XCNT_CUR	DMA34 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310261B8	DMA34_YCNT_CUR	DMA34 Current Row Count (2D only) Register	0x00000000
0x310261C0	DMA34_BWLCNT	DMA34 Bandwidth Limit Count Register	0x00000000
0x310261C4	DMA34_BWLCNT_CUR	DMA34 Bandwidth Limit Count Current Register	0x00000000
0x310261C8	DMA34_BWMCNT	DMA34 Bandwidth Monitor Count Register	0x00000000
0x310261CC	DMA34_BWMCNT_CUR	DMA34 Bandwidth Monitor Count Current Register	0x00000000

Table A-42: ADSP-2156x DMA35 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026100	DMA35_DSCPTR_NXT	DMA35 Pointer to Next Initial Descriptor Register	0x00000000
0x31026104	DMA35_ADDRSTART	DMA35 Start Address of Current Buffer Register	0x00000000
0x31026108	DMA35_CFG	DMA35 Configuration Register	0x00000000
0x3102610C	DMA35_XCNT	DMA35 Inner Loop Count Start Value Register	0x00000000
0x31026110	DMA35_XMOD	DMA35 Inner Loop Address Increment Register	0x00000000
0x31026114	DMA35_YCNT	DMA35 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026118	DMA35_YMOD	DMA35 Outer Loop Address Increment (2D only) Register	0x00000000
0x31026124	DMA35_DSCPTR_CUR	DMA35 Current Descriptor Pointer Register	0x00000000
0x31026128	DMA35_DSCPTR_PRV	DMA35 Previous Initial Descriptor Pointer Register	0x00000000
0x3102612C	DMA35_ADDR_CUR	DMA35 Current Address Register	0x00000000
0x31026130	DMA35_STAT	DMA35 Status Register	0x00006000

Table A-42: ADSP-2156x DMA35 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026134	DMA35_XCNT_CUR	DMA35 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026138	DMA35_YCNT_CUR	DMA35 Current Row Count (2D only) Register	0x00000000
0x31026140	DMA35_BWLCNT	DMA35 Bandwidth Limit Count Register	0x00000000
0x31026144	DMA35_BWLCNT_CUR	DMA35 Bandwidth Limit Count Current Register	0x00000000
0x31026148	DMA35_BWMCNT	DMA35 Bandwidth Monitor Count Register	0x00000000
0x3102614C	DMA35_BWMCNT_CUR	DMA35 Bandwidth Monitor Count Current Register	0x00000000

Table A-43: ADSP-2156x DMA36 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFF080	DMA36_DSCPTR_NXT	DMA36 Pointer to Next Initial Descriptor Register	0x00000000
0x30FFF084	DMA36_ADDRSTART	DMA36 Start Address of Current Buffer Register	0x00000000
0x30FFF088	DMA36_CFG	DMA36 Configuration Register	0x00000000
0x30FFF08C	DMA36_XCNT	DMA36 Inner Loop Count Start Value Register	0x00000000
0x30FFF090	DMA36_XMOD	DMA36 Inner Loop Address Increment Register	0x00000000
0x30FFF094	DMA36_YCNT	DMA36 Outer Loop Count Start Value (2D only) Register	0x00000000
0x30FFF098	DMA36_YMOD	DMA36 Outer Loop Address Increment (2D only) Register	0x00000000
0x30FFF0A4	DMA36_DSCPTR_CUR	DMA36 Current Descriptor Pointer Register	0x00000000
0x30FFF0A8	DMA36_DSCPTR_PRV	DMA36 Previous Initial Descriptor Pointer Register	0x00000000
0x30FFF0AC	DMA36_ADDR_CUR	DMA36 Current Address Register	0x00000000
0x30FFF0B0	DMA36_STAT	DMA36 Status Register	0x00006000
0x30FFF0B4	DMA36_XCNT_CUR	DMA36 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x30FFF0B8	DMA36_YCNT_CUR	DMA36 Current Row Count (2D only) Register	0x00000000
0x30FFF0C0	DMA36_BWLCNT	DMA36 Bandwidth Limit Count Register	0x00000000
0x30FFF0C4	DMA36_BWLCNT_CUR	DMA36 Bandwidth Limit Count Current Register	0x00000000
0x30FFF0C8	DMA36_BWMCNT	DMA36 Bandwidth Monitor Count Register	0x00000000
0x30FFF0CC	DMA36_BWMCNT_CUR	DMA36 Bandwidth Monitor Count Current Register	0x00000000

Table A-44: ADSP-2156x DMA37 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026280	DMA37_DSCPTR_NXT	DMA37 Pointer to Next Initial Descriptor Register	0x00000000
0x31026284	DMA37_ADDRSTART	DMA37 Start Address of Current Buffer Register	0x00000000
0x31026288	DMA37_CFG	DMA37 Configuration Register	0x00000000
0x3102628C	DMA37_XCNT	DMA37 Inner Loop Count Start Value Register	0x00000000
0x31026290	DMA37_XMOD	DMA37 Inner Loop Address Increment Register	0x00000000
0x31026294	DMA37_YCNT	DMA37 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026298	DMA37_YMOD	DMA37 Outer Loop Address Increment (2D only) Register	0x00000000
0x310262A4	DMA37_DSCPTR_CUR	DMA37 Current Descriptor Pointer Register	0x00000000
0x310262A8	DMA37_DSCPTR_PRV	DMA37 Previous Initial Descriptor Pointer Register	0x00000000
0x310262AC	DMA37_ADDR_CUR	DMA37 Current Address Register	0x00000000
0x310262B0	DMA37_STAT	DMA37 Status Register	0x00006000
0x310262B4	DMA37_XCNT_CUR	DMA37 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310262B8	DMA37_YCNT_CUR	DMA37 Current Row Count (2D only) Register	0x00000000
0x310262C0	DMA37_BWLCNT	DMA37 Bandwidth Limit Count Register	0x00000000
0x310262C4	DMA37_BWLCNT_CUR	DMA37 Bandwidth Limit Count Current Register	0x00000000
0x310262C8	DMA37_BWMCNT	DMA37 Bandwidth Monitor Count Register	0x00000000
0x310262CC	DMA37_BWMCNT_CUR	DMA37 Bandwidth Monitor Count Current Register	0x00000000

Table A-45: ADSP-2156x DMA38 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026200	DMA38_DSCPTR_NXT	DMA38 Pointer to Next Initial Descriptor Register	0x00000000
0x31026204	DMA38_ADDRSTART	DMA38 Start Address of Current Buffer Register	0x00000000
0x31026208	DMA38_CFG	DMA38 Configuration Register	0x00000000
0x3102620C	DMA38_XCNT	DMA38 Inner Loop Count Start Value Register	0x00000000
0x31026210	DMA38_XMOD	DMA38 Inner Loop Address Increment Register	0x00000000
0x31026214	DMA38_YCNT	DMA38 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026218	DMA38_YMOD	DMA38 Outer Loop Address Increment (2D only) Register	0x00000000

Table A-45: ADSP-2156x DMA38 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31026224	DMA38_DSCPTR_CUR	DMA38 Current Descriptor Pointer Register	0x00000000
0x31026228	DMA38_DSCPTR_PRV	DMA38 Previous Initial Descriptor Pointer Register	0x00000000
0x3102622C	DMA38_ADDR_CUR	DMA38 Current Address Register	0x00000000
0x31026230	DMA38_STAT	DMA38 Status Register	0x00006000
0x31026234	DMA38_XCNT_CUR	DMA38 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026238	DMA38_YCNT_CUR	DMA38 Current Row Count (2D only) Register	0x00000000
0x31026240	DMA38_BWLCNT	DMA38 Bandwidth Limit Count Register	0x00000000
0x31026244	DMA38_BWLCNT_CUR	DMA38 Bandwidth Limit Count Current Register	0x00000000
0x31026248	DMA38_BWMCNT	DMA38 Bandwidth Monitor Count Register	0x00000000
0x3102624C	DMA38_BWMCNT_CUR	DMA38 Bandwidth Monitor Count Current Register	0x00000000

Table A-46: ADSP-2156x DMA39 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109A000	DMA39_DSCPTR_NXT	DMA39 Pointer to Next Initial Descriptor Register	0x00000000
0x3109A004	DMA39_ADDRSTART	DMA39 Start Address of Current Buffer Register	0x00000000
0x3109A008	DMA39_CFG	DMA39 Configuration Register	0x00000220
0x3109A00C	DMA39_XCNT	DMA39 Inner Loop Count Start Value Register	0x00000000
0x3109A010	DMA39_XMOD	DMA39 Inner Loop Address Increment Register	0x00000000
0x3109A014	DMA39_YCNT	DMA39 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109A018	DMA39_YMOD	DMA39 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109A024	DMA39_DSCPTR_CUR	DMA39 Current Descriptor Pointer Register	0x00000000
0x3109A028	DMA39_DSCPTR_PRV	DMA39 Previous Initial Descriptor Pointer Register	0x00000000
0x3109A02C	DMA39_ADDR_CUR	DMA39 Current Address Register	0x00000000
0x3109A030	DMA39_STAT	DMA39 Status Register	0x00006000
0x3109A034	DMA39_XCNT_CUR	DMA39 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109A038	DMA39_YCNT_CUR	DMA39 Current Row Count (2D only) Register	0x00000000
0x3109A040	DMA39_BWLCNT	DMA39 Bandwidth Limit Count Register	0x00000000
0x3109A044	DMA39_BWLCNT_CUR	DMA39 Bandwidth Limit Count Current Register	0x00000000

Table A-46: ADSP-2156x DMA39 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109A048	DMA39_BWMCNT	DMA39 Bandwidth Monitor Count Register	0x00000000
0x3109A04C	DMA39_BWMCNT_CUR	DMA39 Bandwidth Monitor Count Current Register	0x00000000

Table A-47: ADSP-2156x DMA4 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31022200	DMA4_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x31022204	DMA4_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000
0x31022208	DMA4_CFG	DMA4 Configuration Register	0x00000000
0x3102220C	DMA4_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x31022210	DMA4_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x31022214	DMA4_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022218	DMA4_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022224	DMA4_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x31022228	DMA4_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x3102222C	DMA4_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x31022230	DMA4_STAT	DMA4 Status Register	0x00006000
0x31022234	DMA4_XCNT_CUR	DMA4 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022238	DMA4_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000
0x31022240	DMA4_BWLCNT	DMA4 Bandwidth Limit Count Register	0x00000000
0x31022244	DMA4_BWLCNT_CUR	DMA4 Bandwidth Limit Count Current Register	0x00000000
0x31022248	DMA4_BWMCNT	DMA4 Bandwidth Monitor Count Register	0x00000000
0x3102224C	DMA4_BWMCNT_CUR	DMA4 Bandwidth Monitor Count Current Register	0x00000000

Table A-48: ADSP-2156x DMA40 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109A080	DMA40_DSCPTR_NXT	DMA40 Pointer to Next Initial Descriptor Register	0x00000000
0x3109A084	DMA40_ADDRSTART	DMA40 Start Address of Current Buffer Register	0x00000000
0x3109A088	DMA40_CFG	DMA40 Configuration Register	0x00000222
0x3109A08C	DMA40_XCNT	DMA40 Inner Loop Count Start Value Register	0x00000000

Table A-48: ADSP-2156x DMA40 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109A090	DMA40_XMOD	DMA40 Inner Loop Address Increment Register	0x00000000
0x3109A094	DMA40_YCNT	DMA40 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109A098	DMA40_YMOD	DMA40 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109A0A4	DMA40_DSCPTR_CUR	DMA40 Current Descriptor Pointer Register	0x00000000
0x3109A0A8	DMA40_DSCPTR_PRV	DMA40 Previous Initial Descriptor Pointer Register	0x00000000
0x3109A0AC	DMA40_ADDR_CUR	DMA40 Current Address Register	0x00000000
0x3109A0B0	DMA40_STAT	DMA40 Status Register	0x00006000
0x3109A0B4	DMA40_XCNT_CUR	DMA40 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109A0B8	DMA40_YCNT_CUR	DMA40 Current Row Count (2D only) Register	0x00000000
0x3109A0C0	DMA40_BWLCNT	DMA40 Bandwidth Limit Count Register	0x00000000
0x3109A0C4	DMA40_BWLCNT_CUR	DMA40 Bandwidth Limit Count Current Register	0x00000000
0x3109A0C8	DMA40_BWMCNT	DMA40 Bandwidth Monitor Count Register	0x00000000
0x3109A0CC	DMA40_BWMCNT_CUR	DMA40 Bandwidth Monitor Count Current Register	0x00000000

Table A-49: ADSP-2156x DMA43 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109B000	DMA43_DSCPTR_NXT	DMA43 Pointer to Next Initial Descriptor Register	0x00000000
0x3109B004	DMA43_ADDRSTART	DMA43 Start Address of Current Buffer Register	0x00000000
0x3109B008	DMA43_CFG	DMA43 Configuration Register	0x00000220
0x3109B00C	DMA43_XCNT	DMA43 Inner Loop Count Start Value Register	0x00000000
0x3109B010	DMA43_XMOD	DMA43 Inner Loop Address Increment Register	0x00000000
0x3109B014	DMA43_YCNT	DMA43 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109B018	DMA43_YMOD	DMA43 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109B024	DMA43_DSCPTR_CUR	DMA43 Current Descriptor Pointer Register	0x00000000
0x3109B028	DMA43_DSCPTR_PRV	DMA43 Previous Initial Descriptor Pointer Register	0x00000000
0x3109B02C	DMA43_ADDR_CUR	DMA43 Current Address Register	0x00000000
0x3109B030	DMA43_STAT	DMA43 Status Register	0x0000B000

Table A-49: ADSP-2156x DMA43 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109B034	DMA43_XCNT_CUR	DMA43 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109B038	DMA43_YCNT_CUR	DMA43 Current Row Count (2D only) Register	0x00000000
0x3109B040	DMA43_BWLCNT	DMA43 Bandwidth Limit Count Register	0x00000000
0x3109B044	DMA43_BWLCNT_CUR	DMA43 Bandwidth Limit Count Current Register	0x00000000
0x3109B048	DMA43_BWMCNT	DMA43 Bandwidth Monitor Count Register	0x00000000
0x3109B04C	DMA43_BWMCNT_CUR	DMA43 Bandwidth Monitor Count Current Register	0x00000000

Table A-50: ADSP-2156x DMA44 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109B080	DMA44_DSCPTR_NXT	DMA44 Pointer to Next Initial Descriptor Register	0x00000000
0x3109B084	DMA44_ADDRSTART	DMA44 Start Address of Current Buffer Register	0x00000000
0x3109B088	DMA44_CFG	DMA44 Configuration Register	0x00000222
0x3109B08C	DMA44_XCNT	DMA44 Inner Loop Count Start Value Register	0x00000000
0x3109B090	DMA44_XMOD	DMA44 Inner Loop Address Increment Register	0x00000000
0x3109B094	DMA44_YCNT	DMA44 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109B098	DMA44_YMOD	DMA44 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109B0A4	DMA44_DSCPTR_CUR	DMA44 Current Descriptor Pointer Register	0x00000000
0x3109B0A8	DMA44_DSCPTR_PRV	DMA44 Previous Initial Descriptor Pointer Register	0x00000000
0x3109B0AC	DMA44_ADDR_CUR	DMA44 Current Address Register	0x00000000
0x3109B0B0	DMA44_STAT	DMA44 Status Register	0x0000B000
0x3109B0B4	DMA44_XCNT_CUR	DMA44 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109B0B8	DMA44_YCNT_CUR	DMA44 Current Row Count (2D only) Register	0x00000000
0x3109B0C0	DMA44_BWLCNT	DMA44 Bandwidth Limit Count Register	0x00000000
0x3109B0C4	DMA44_BWLCNT_CUR	DMA44 Bandwidth Limit Count Current Register	0x00000000
0x3109B0C8	DMA44_BWMCNT	DMA44 Bandwidth Monitor Count Register	0x00000000
0x3109B0CC	DMA44_BWMCNT_CUR	DMA44 Bandwidth Monitor Count Current Register	0x00000000

Table A-51: ADSP-2156x DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022280	DMA5_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000
0x31022284	DMA5_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x31022288	DMA5_CFG	DMA5 Configuration Register	0x00000000
0x3102228C	DMA5_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x31022290	DMA5_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000
0x31022294	DMA5_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022298	DMA5_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x310222A4	DMA5_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x310222A8	DMA5_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x310222AC	DMA5_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x310222B0	DMA5_STAT	DMA5 Status Register	0x00006000
0x310222B4	DMA5_XCNT_CUR	DMA5 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310222B8	DMA5_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000
0x310222C0	DMA5_BWLCNT	DMA5 Bandwidth Limit Count Register	0x00000000
0x310222C4	DMA5_BWLCNT_CUR	DMA5 Bandwidth Limit Count Current Register	0x00000000
0x310222C8	DMA5_BWMCNT	DMA5 Bandwidth Monitor Count Register	0x00000000
0x310222CC	DMA5_BWMCNT_CUR	DMA5 Bandwidth Monitor Count Current Register	0x00000000

Table A-52: ADSP-2156x DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022300	DMA6_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x31022304	DMA6_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x31022308	DMA6_CFG	DMA6 Configuration Register	0x00000000
0x3102230C	DMA6_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000
0x31022310	DMA6_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x31022314	DMA6_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022318	DMA6_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022324	DMA6_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x31022328	DMA6_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x3102232C	DMA6_ADDR_CUR	DMA6 Current Address Register	0x00000000

Table A-52: ADSP-2156x DMA6 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31022330	DMA6_STAT	DMA6 Status Register	0x00006000
0x31022334	DMA6_XCNT_CUR	DMA6 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022338	DMA6_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000
0x31022340	DMA6_BWLCNT	DMA6 Bandwidth Limit Count Register	0x00000000
0x31022344	DMA6_BWLCNT_CUR	DMA6 Bandwidth Limit Count Current Register	0x00000000
0x31022348	DMA6_BWMCNT	DMA6 Bandwidth Monitor Count Register	0x00000000
0x3102234C	DMA6_BWMCNT_CUR	DMA6 Bandwidth Monitor Count Current Register	0x00000000

Table A-53: ADSP-2156x DMA7 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31022380	DMA7_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x31022384	DMA7_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000
0x31022388	DMA7_CFG	DMA7 Configuration Register	0x00000000
0x3102238C	DMA7_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x31022390	DMA7_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x31022394	DMA7_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022398	DMA7_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x310223A4	DMA7_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x310223A8	DMA7_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x310223AC	DMA7_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x310223B0	DMA7_STAT	DMA7 Status Register	0x00006000
0x310223B4	DMA7_XCNT_CUR	DMA7 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310223B8	DMA7_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000
0x310223C0	DMA7_BWLCNT	DMA7 Bandwidth Limit Count Register	0x00000000
0x310223C4	DMA7_BWLCNT_CUR	DMA7 Bandwidth Limit Count Current Register	0x00000000
0x310223C8	DMA7_BWMCNT	DMA7 Bandwidth Monitor Count Register	0x00000000
0x310223CC	DMA7_BWMCNT_CUR	DMA7 Bandwidth Monitor Count Current Register	0x00000000

Table A-54: ADSP-2156x DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A7000	DMA8_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000
0x310A7004	DMA8_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x310A7008	DMA8_CFG	DMA8 Configuration Register	0x00000000
0x310A700C	DMA8_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x310A7010	DMA8_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x310A7014	DMA8_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x310A7018	DMA8_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x310A7024	DMA8_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x310A7028	DMA8_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x310A702C	DMA8_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x310A7030	DMA8_STAT	DMA8 Status Register	0x00006000
0x310A7034	DMA8_XCNT_CUR	DMA8 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310A7038	DMA8_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000
0x310A7040	DMA8_BWLCNT	DMA8 Bandwidth Limit Count Register	0x00000000
0x310A7044	DMA8_BWLCNT_CUR	DMA8 Bandwidth Limit Count Current Register	0x00000000
0x310A7048	DMA8_BWMCNT	DMA8 Bandwidth Monitor Count Register	0x00000000
0x310A704C	DMA8_BWMCNT_CUR	DMA8 Bandwidth Monitor Count Current Register	0x00000000

Table A-55: ADSP-2156x DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A7080	DMA9_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x310A7084	DMA9_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x310A7088	DMA9_CFG	DMA9 Configuration Register	0x00000000
0x310A708C	DMA9_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x310A7090	DMA9_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x310A7094	DMA9_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x310A7098	DMA9_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x310A70A4	DMA9_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x310A70A8	DMA9_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x310A70AC	DMA9_ADDR_CUR	DMA9 Current Address Register	0x00000000

Table A-55: ADSP-2156x DMA9 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A70B0	DMA9_STAT	DMA9 Status Register	0x00006000
0x310A70B4	DMA9_XCNT_CUR	DMA9 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310A70B8	DMA9_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000
0x310A70C0	DMA9_BWLCNT	DMA9 Bandwidth Limit Count Register	0x00000000
0x310A70C4	DMA9_BWLCNT_CUR	DMA9 Bandwidth Limit Count Current Register	0x00000000
0x310A70C8	DMA9_BWMCNT	DMA9 Bandwidth Monitor Count Register	0x00000000
0x310A70CC	DMA9_BWMCNT_CUR	DMA9 Bandwidth Monitor Count Current Register	0x00000000

Table A-56: ADSP-2156x DMC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31070004	DMC0_CTL	DMC0 Control Register	0x00000000
0x31070008	DMC0_STAT	DMC0 Status Register	0x00000001
0x3107000C	DMC0_EFFCTL	DMC0 Efficiency Control Register	0x00440000
0x31070010	DMC0_PRI0	DMC0 Priority ID Register 1	0x00000000
0x31070014	DMC0_PRIOMSK	DMC0 Priority ID Mask Register 1	0x00000000
0x31070018	DMC0_PRI02	DMC0 Priority ID Register 2	0x00000000
0x3107001C	DMC0_PRIOMSK2	DMC0 Priority ID Mask Register 2	0x00000000
0x31070040	DMC0_CFG	DMC0 Configuration Register	0x00000000
0x31070044	DMC0_TR0	DMC0 Timing 0 Register	0x00000000
0x31070048	DMC0_TR1	DMC0 Timing 1 Register	0x00000000
0x3107004C	DMC0_TR2	DMC0 Timing 2 Register	0x00000000
0x3107005C	DMC0_MSK	DMC0 Mask (Mode Register Shadow) Register	0x00000000
0x31070060	DMC0_MR	DMC0 Shadow MR0 Register (DDR3)	0x00000000
0x31070064	DMC0_MR1	DMC0 Shadow MR1 Register (DDR3)	0x00000000
0x31070068	DMC0_MR2	DMC0 Shadow MR2 Register (DDR3)	0x00000000
0x3107006C	DMC0_EMR3	DMC0 Shadow EMR3 Register	0x00000000
0x31070080	DMC0_DLLCTL	DMC0 DLL Control Register	0x0000054B
0x31070090	DMC0_DT_CALIB_ADDR	DMC0 Data Calibration Address Register	0x00000000
0x31070094	DMC0_DT_DATA_CAL- IB_DATA0	DMC0 Data Calibration Data 0 Register	0x00000000

Table A-56: ADSP-2156x DMC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31070098	DMC0_DT_DATA_CAL-IB_DATA1	DMC0 Data Calibration Data 1 Register	0xFFFFFFFF
0x31070100	DMC0_RDDATABUFID1	DMC0 DMC Read Data Buffer ID Register 1	0x00000000
0x31070104	DMC0_RDDATABUFMSK1	DMC0 DMC Read Data Buffer Mask Register 1	0x00000000
0x31070108	DMC0_RDDATABUFID2	DMC0 DMC Read Data Buffer ID Register 2	0x00000000
0x3107010C	DMC0_RDDATABUFMSK2	DMC0 DMC Read Data Buffer Mask Register 2	0x00000000

Table A-57: ADSP-2156x DPM0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31090000	DPM0_CTL	DPM0 Control Register	0x00000000
0x31090004	DPM0_STAT	DPM0 Status Register	0x00000001
0x31090070	DPM0_PER_DIS0	DPM0 Peripherals Disable Register 0	0x00000000
0x31090074	DPM0_PER_DIS1	DPM0 Peripherals Disable Register 1	0x00000000
0x31090084	DPM0_REVID	DPM0 Revision ID	0x00000020

Table A-58: ADSP-2156x EMDMA0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310E002C	EMDMA0_CTL	EMDMA0 External Memory DMA Control Register	0x00000000
0x310E0080	EMDMA0_INDX1	EMDMA0 External Index Register	0x00000000
0x310E0084	EMDMA0_MOD1	EMDMA0 External Modifier Register	0x00000000
0x310E0088	EMDMA0_CNT1	EMDMA0 External Count Register	0x00000000
0x310E008C	EMDMA0_INDX0	EMDMA0 Internal Index Register	0x00000000
0x310E0090	EMDMA0_MOD0	EMDMA0 Internal Modifier Register	0x00000000
0x310E0094	EMDMA0_CNT0	EMDMA0 Internal Count Register	0x00000000
0x310E0098	EMDMA0_CHNPTR	EMDMA0 Chain Pointer Register	0x00000000
0x310E009C	EMDMA0_BASE	EMDMA0 External Base Address Register	0x00000000
0x310E00A0	EMDMA0_TPTR	EMDMA0 Tap List Pointer Register	0x00000000
0x310E00A4	EMDMA0_BUFLEN	EMDMA0 Circular Buffer Length Register	0x00000000
0x310E00AC	EMDMA0_TCNT	EMDMA0 Delay Line Tap Count Register	0x00000000

Table A-59: ADSP-2156x EMDMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310E0030	EMDMA1_CTL	EMDMA1 External Memory DMA Control Register	0x00000000
0x310E00C0	EMDMA1_INDX1	EMDMA1 External Index Register	0x00000000
0x310E00C4	EMDMA1_MOD1	EMDMA1 External Modifier Register	0x00000000
0x310E00C8	EMDMA1_CNT1	EMDMA1 External Count Register	0x00000000
0x310E00CC	EMDMA1_INDX0	EMDMA1 Internal Index Register	0x00000000
0x310E00D0	EMDMA1_MOD0	EMDMA1 Internal Modifier Register	0x00000000
0x310E00D4	EMDMA1_CNT0	EMDMA1 Internal Count Register	0x00000000
0x310E00D8	EMDMA1_CHNPTR	EMDMA1 Chain Pointer Register	0x00000000
0x310E00DC	EMDMA1_BASE	EMDMA1 External Base Address Register	0x00000000
0x310E00E0	EMDMA1_TPTR	EMDMA1 Tap List Pointer Register	0x00000000
0x310E00E4	EMDMA1_BUFLEN	EMDMA1 Circular Buffer Length Register	0x00000000
0x310E00EC	EMDMA1_TCNT	EMDMA1 Delay Line Tap Count Register	0x00000000

Table A-60: ADSP-2156x FIR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C3000	FIR0_CTL1	FIR0 FIR Global Control Register	0x00000000
0x310C3004	FIR0_DMASTAT	FIR0 FIR DMA Status Register	0x00000000
0x310C3008	FIR0_MACSTAT	FIR0 FIR MAC Status Register	0x00000000
0x310C3010	FIR0_DBG_CTL	FIR0 FIR Debug Control Register	0x00000000
0x310C3014	FIR0_DBG_ADDR	FIR0 Debug Address Register	0x00000000
0x310C3018	FIR0_DBG_WRDAT	FIR0 FIR Debug Data Write Register	0x00000000
0x310C301C	FIR0_DBG_RDDAT	FIR0 FIR Debug Data Read Register	0x00000000
0x310C3040	FIR0_CTL2	FIR0 FIR Channel Control Register	0x00000000
0x310C3044	FIR0_INIDX	FIR0 FIR Input Data Index Register	0x00000000
0x310C3048	FIR0_INMOD	FIR0 FIR Input Data Modifier Register	0x00000000
0x310C304C	FIR0_INCNT	FIR0 FIR Input Data Count Register	0x00000000
0x310C3050	FIR0_INBASE	FIR0 FIR Input Data Base Register	0x00000000
0x310C3054	FIR0_OUTIDX	FIR0 FIR Output Data Index Register	0x00000000
0x310C3058	FIR0_OUTMOD	FIR0 FIR Output Data Modifier Register	0x00000000
0x310C305C	FIR0_OUTCNT	FIR0 FIR Output Data Count Register	0x00000000

Table A-60: ADSP-2156x FIR0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C3060	FIR0_OUTBASE	FIR0 FIR Output Data Base Register	0x00000000
0x310C3064	FIR0_COEFIDX	FIR0 FIR Coefficient Index Register	0x00000000
0x310C3068	FIR0_COEFMOD	FIR0 FIR Coefficient Modifier Register	0x00000000
0x310C306C	FIR0_COEFCNT	FIR0 FIR Coefficient Count Register	0x00000000
0x310C3070	FIR0_CHNPTR	FIR0 FIR Chain Pointer Register	0x00000000
0x310C3074	FIR0_SCTL1	FIR0 software control register 1	0x00000000
0x310C3078	FIR0_SCTL2	FIR0 software control register 2	0x00000000
0x310C307C	FIR0_SGCTL	FIR0 Secondary global control register	0x00000000

Table A-61: ADSP-2156x HADC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31016000	HADC0_CTL	HADC0 Control Register	0x000001AA
0x31016004	HADC0_CHAN_MSK	HADC0 Channel Mask Register	0x0000FF00
0x31016008	HADC0_IMSK	HADC0 Interrupt Mask Register	0x00000000
0x3101600C	HADC0_STAT	HADC0 Status Register	0x00000000
0x31016010	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31016014	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31016018	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x3101601C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000

Table A-62: ADSP-2156x IIR0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C4000	IIR0_CTL1	IIR0 Global Control Register	0x00000000
0x310C4004	IIR0_DMASTAT	IIR0 DMA Status Register	0x00000000
0x310C4008	IIR0_MACSTAT	IIR0 MAC Status Register	0x00000000
0x310C400C	IIR0_DBG_CTL	IIR0 IIR Debug Control Register	0x00000000
0x310C4010	IIR0_DBG_ADDR	IIR0 IIR Debug Address Register	0x00000000
0x310C4014	IIR0_DBG_WRDAT_LO	IIR0 IIR Debug Write Data Low Register	0x00000000
0x310C4018	IIR0_DBG_WRDAT_HI	IIR0 IIR Debug Write Data High Register	0x00000000
0x310C401C	IIR0_DBG_RDDAT_LO	IIR0 IIR Debug Read Data Low Register	0x00000000

Table A-62: ADSP-2156x IIR0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C4020	IIR0_DBG_RDDAT_HI	IIR0 IIR Debug Read Data High Register	0x00000000
0x310C4040	IIR0_CTL2	IIR0 Channel Control Register	0x00000000
0x310C4044	IIR0_INIDX	IIR0 Input Data Index Register	0x00000000
0x310C4048	IIR0_INMOD	IIR0 Input Data Index Modifier Register	0x00000000
0x310C404C	IIR0_INLEN	IIR0 Input Data Buffer Length Register	0x00000000
0x310C4050	IIR0_INBASE	IIR0 Input Buffer Base Register	0x00000000
0x310C4054	IIR0_OUTIDX	IIR0 Output Data Buffer Index Register	0x00000000
0x310C4058	IIR0_OUTMOD	IIR0 IIR Output Data Index Modifier Register	0x00000000
0x310C405C	IIR0_OUTLEN	IIR0 IIR Output Data Buffer Length Register	0x00000000
0x310C4060	IIR0_OUTBASE	IIR0 Output Buffer Base Register	0x00000000
0x310C4064	IIR0_COEFIDX	IIR0 Coefficient Buffer Index Register	0x00000000
0x310C4068	IIR0_COEFMOD	IIR0 Coefficient Index Modifier Register	0x00000000
0x310C406C	IIR0_COEFLEN	IIR0 Coefficient Buffer Length Register	0x00000000
0x310C4070	IIR0_CHNPTR	IIR0 Chain Pointer Register	0x00000000
0x310C4074	IIR0_SCTL1	IIR0 Software Control Register1	0x00000000
0x310C4078	IIR0_SCTL2	IIR0 Software Control Register2	0x00000000
0x310C407C	IIR0_SGCTL	IIR0 Secondary Global Control Register	0x00000000

Table A-63: ADSP-2156x L2CTL0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31080000	L2CTL0_CTL	L2CTL0 Control Register	0x00000000
0x31080010	L2CTL0_STAT	L2CTL0 Status Register	0x00000000
0x31080014	L2CTL0_RPCR	L2CTL0 Read Priority Count Register	0x00000F0F
0x31080018	L2CTL0_WPCR	L2CTL0 Write Priority Count Register	0x00000F0F
0x31080024	L2CTL0_INIT	L2CTL0 Initialization Register	0x00000000
0x31080038	L2CTL0_ISTAT	L2CTL0 Initialization Status Register	0x00000000
0x3108003C	L2CTL0_PCTL	L2CTL0 Power Control Register	0x00000000
0x31080040	L2CTL0_ERRADDR0	L2CTL0 ECC Error Address 0 Register	0x20000000
0x31080044	L2CTL0_ERRADDR1	L2CTL0 ECC Error Address 1 Register	0x20020000
0x31080048	L2CTL0_ERRADDR2	L2CTL0 ECC Error Address 2 Register	0x20040000

Table A-63: ADSP-2156x L2CTL0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108004C	L2CTL0_ERRADDR3	L2CTL0 ECC Error Address 3 Register	0x20060000
0x31080050	L2CTL0_ERRADDR4	L2CTL0 ECC Error Address 4 Register	0x20080000
0x31080054	L2CTL0_ERRADDR5	L2CTL0 ECC Error Address 5 Register	0x200A0000
0x31080058	L2CTL0_ERRADDR6	L2CTL0 ECC Error Address 6 Register	0x200C0000
0x3108005C	L2CTL0_ERRADDR7	L2CTL0 ECC Error Address 7 Register	0x200E0000
0x31080060	L2CTL0_ERRADDR8	L2CTL0 ECC Error Address 8 Register	0x20100000
0x31080080	L2CTL0_ET0	L2CTL0 Error Type 0 Register	0x00000000
0x31080084	L2CTL0_EADDR0	L2CTL0 Error Type 0 Address Register	0x20000000
0x31080088	L2CTL0_ET1	L2CTL0 Error Type 1 Register	0x00000000
0x3108008C	L2CTL0_EADDR1	L2CTL0 Error Type 1 Address Register	0x20000000
0x310800EC	L2CTL0_SCTL	L2CTL0 Scrub Control Register	0x00000000
0x310800F0	L2CTL0_SADR	L2CTL0 Scrub Start Address Register	0x00000000
0x310800F4	L2CTL0_SCNT	L2CTL0 Scrub Count Register	0x00000000
0x310800FC	L2CTL0_REVID	L2CTL0 Revision ID Register	0x00000001

Table A-64: ADSP-2156x LP0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30FFE000	LP0_CTL	LP0 Control Register	0x00000000
0x30FFE004	LP0_STAT	LP0 Status Register	0x00000000
0x30FFE008	LP0_DIV	LP0 Clock Divider Value Register	0x00000000
0x30FFE010	LP0_TX	LP0 Transmit Buffer Register	0x00000000
0x30FFE014	LP0_RX	LP0 Receive Buffer Register	0x00000000
0x30FFE018	LP0_TXIN_SHDW	LP0 Shadow Input Transmit Buffer Register	0x00000000
0x30FFE01C	LP0_TXOUT_SHDW	LP0 Shadow Output Transmit Buffer Register	0x00000000

Table A-65: ADSP-2156x LP1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30FFE100	LP1_CTL	LP1 Control Register	0x00000000
0x30FFE104	LP1_STAT	LP1 Status Register	0x00000000
0x30FFE108	LP1_DIV	LP1 Clock Divider Value Register	0x00000000

Table A-65: ADSP-2156x LP1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x30FFE110	LP1_TX	LP1 Transmit Buffer Register	0x00000000
0x30FFE114	LP1_RX	LP1 Receive Buffer Register	0x00000000
0x30FFE118	LP1_TXIN_SHDW	LP1 Shadow Input Transmit Buffer Register	0x00000000
0x30FFE11C	LP1_TXOUT_SHDW	LP1 Shadow Output Transmit Buffer Register	0x00000000

Table A-66: ADSP-2156x MEC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A3000	MEC0_PEIRQ_GCTL[p]	MEC0 Parity Error Interrupt Request Global Control Register	0x00000000
0x310A3010	MEC0_PEIRQ_GSTAT[p]	MEC0 Parity Error Interrupt Request Global Status Register	0x00000000
0x310A3040	MEC0_PERR_CTL	MEC0 Parity Error Control Register	0x00000000
0x310A3080	MEC0_PERR_STAT	MEC0 Parity Error Status Register	0x00000000
0x310A30C0	MEC0_PERR_IMASK	MEC0 Parity Error Interrupt Mask Register	0x00000000
0x310A3100	MEC0_EEIRQ_GCTL[q]	MEC0 ECC Error Interrupt Request Global Control Register	0x00000000
0x310A3110	MEC0_EEIRQ_GSTAT[q]	MEC0 ECC Error Interrupt Request Global Status Register	0x00000000
0x310A3140	MEC0_ECCERR_CTL[y]	MEC0 ECC Error Control Register	0x00000000
0x310A3180	MEC0_ECCERR_STAT[y]	MEC0 ECC Error Status Register	0x00000000
0x310A31C0	MEC0_ECCERR_IMASK[y]	MEC0 ECC Error Interrupt Mask Register	0x00000000
0x310A3F00	MEC0_CLR	MEC0 Clear Register	0x00000000
0x310A3FD0	MEC0_PID4	MEC0 Peripheral ID4 Register	0x00000000
0x310A3FD4	MEC0_PID5	MEC0 Peripheral ID5 Register	0x00000000
0x310A3FD8	MEC0_PID6	MEC0 Peripheral ID6 Register	0x00000000
0x310A3FDC	MEC0_PID7	MEC0 Peripheral ID7 Register	0x00000000
0x310A3FE0	MEC0_PID0	MEC0 Peripheral ID0 Register	0x00000003
0x310A3FE4	MEC0_PID1	MEC0 Peripheral ID1 Register	0x00000050
0x310A3FE8	MEC0_PID2	MEC0 Peripheral ID2 Register	0x0000000E
0x310A3FEC	MEC0_PID3	MEC0 Peripheral ID3 Register	0x00000000
0x310A3FF0	MEC0_CID0	MEC0 Component ID0 Register	0x0000000D
0x310A3FF4	MEC0_CID1	MEC0 Component ID1 Register	0x000000F0

Table A-66: ADSP-2156x MEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A3FF8	MEC0_CID2	MEC0 Component ID2 Register	0x00000005
0x310A3FFC	MEC0_CID3	MEC0 Component ID3 Register	0x000000B1

Table A-67: ADSP-2156x MISCREG MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A9000	MIS-CREG_PFB_RANGE_SELECT	MISCREG Prefetch Range Selection Register	0xFFFFFFFF

Table A-68: ADSP-2156x MLB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109D000	MLB0_CTL0	MLB0 MediaLB Control 0 Register	0x00000000
0x3109D00C	MLB0_MS0	MLB0 Channel Status 0 Register	0x00000000
0x3109D014	MLB0_MS1	MLB0 Channel Status 1 Register	0x00000000
0x3109D020	MLB0_MSS	MLB0 System Status Register	0x00000000
0x3109D024	MLB0_MSD	MLB0 System Data Register	0x00000000
0x3109D02C	MLB0_MIEN	MLB0 Interrupt Enable Register	0x00000000
0x3109D034	MLB0_GCTL	MLB0 MLB Global Control Register	0x00000000
0x3109D03C	MLB0_CTL1	MLB0 Control 1 Register	0x00000000
0x3109D080	MLB0_HCTL	MLB0 HBI Control Register	0x00000000
0x3109D088	MLB0_HCMR0	MLB0 HBI Channel Mask 0 Register	0x00000000
0x3109D08C	MLB0_HCMR1	MLB0 HBI Channel Mask 1 Register	0x00000000
0x3109D090	MLB0_HCER0	MLB0 HBI Channel Error 0 Register	0x00000000
0x3109D094	MLB0_HCER1	MLB0 HBI Channel Error 1 Register	0x00000000
0x3109D098	MLB0_HCBR0	MLB0 HBI Channel Busy 0 Register	0x00000000
0x3109D09C	MLB0_HCBR1	MLB0 HBI Channel Busy 1 Register	0x00000000
0x3109D0C0	MLB0_MDAT0	MLB0 Memory Interface Control Data 0 Register	0x00000000
0x3109D0C4	MLB0_MDAT1	MLB0 Memory Interface Control Data 1 Register	0x00000000
0x3109D0C8	MLB0_MDAT2	MLB0 Memory Interface Control Data 2 Register	0x00000000
0x3109D0CC	MLB0_MDAT3	MLB0 Memory Interface Control Data 3 Register	0x00000000

Table A-68: ADSP-2156x MLB0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109D0D0	MLB0_MDWE0	MLB0 Memory Interface Control Data Write Enable 0 Register	0x00000000
0x3109D0D4	MLB0_MDWE1	MLB0 Memory Interface Control Data Write Enable 1 Register	0x00000000
0x3109D0D8	MLB0_MDWE2	MLB0 Memory Interface Control Data Write Enable 2 Register	0x00000000
0x3109D0DC	MLB0_MDWE3	MLB0 Memory Interface Control Data Write Enable 3 Register	0x00000000
0x3109D0E0	MLB0_MCTL	MLB0 Memory Interface Control Register	0x00000000
0x3109D0E4	MLB0_MADR	MLB0 Memory Interface Address Register	0x00000000
0x3109D3C0	MLB0_ACTL	MLB0 Bus Control Register	0x00000000
0x3109D3D0	MLB0_ACSR0	MLB0 Peripheral Channel Status 0 Register	0x00000000
0x3109D3D4	MLB0_ACSR1	MLB0 Peripheral Channel Status 1 Register	0x00000000
0x3109D3D8	MLB0_ACMR0	MLB0 Peripheral Channel Mask 0 Register	0x00000000
0x3109D3DC	MLB0_ACMR1	MLB0 Peripheral Channel Mask 1 Register	0x00000000

Table A-69: ADSP-2156x SCB3 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30105020	SCB3_APB_DCLK_0_IB_SY NC_MODE	SCB3 APB DCLK 0 Synchronization Mode Register	0x00000004

Table A-70: ADSP-2156x OSPI0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31027000	OSPI0_CTL	OSPI0 Octal SPI Control Register	0x80780081
0x31027004	OSPI0_DRICTL	OSPI0 Device Read Instruction Control Register	0x00000003
0x31027008	OSPI0_DWICTL	OSPI0 Device Write Instruction Control Register	0x00000002
0x3102700C	OSPI0_DLY	OSPI0 Device Delay Register	0x00000000
0x31027010	OSPI0_RDC	OSPI0 Read Data Capture Register	0x00000001
0x31027014	OSPI0_DSCTL	OSPI0 Device Size Control Register	0x00101002
0x31027024	OSPI0_REMAPADDR	OSPI0 Remap Address Register	0x00000000
0x31027028	OSPI0_MBCTL	OSPI0 Mode Bit Control Register	0x00000000
0x31027038	OSPI0_WCCTL	OSPI0 Write Completion Control Register	0x00010005

Table A-70: ADSP-2156x OSPI0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3102703C	OSPI0_POLLEXP	OSPI0 Polling Expiration Register	0xFFFFFFFF
0x31027040	OSPI0_ISTAT	OSPI0 Interrupt Status Register	0x00000000
0x31027044	OSPI0_IMSK	OSPI0 Interrupt Mask Register	0x00000000
0x31027050	OSPI0_WRPROT_LWR	OSPI0 Lower Write Protection Register	0x00000000
0x31027054	OSPI0_WRPROT_UP	OSPI0 Upper Write Protection Register	0x00000000
0x31027058	OSPI0_WRPROT_CTL	OSPI0 Write Protection Control Register	0x00000000
0x3102708C	OSPI0_FCMCTL	OSPI0 Flash Command Control Memory Register	0x00000000
0x31027090	OSPI0_FCCTL	OSPI0 Flash Command Control Register	0x00000000
0x31027094	OSPI0_FCA	OSPI0 Flash Command Address Register	0x00000000
0x310270A0	OSPI0_FCRD_LWR	OSPI0 Flash Command Read Data Register (Lower)	0x00000000
0x310270A4	OSPI0_FCRD_UP	OSPI0 Flash Command Read Data Register (Upper)	0x00000000
0x310270A8	OSPI0_FCWD_LWR	OSPI0 Flash Command Write Data Register (Lower)	0x00000000
0x310270AC	OSPI0_FCWD_UP	OSPI0 Flash Command Write Data Register (Upper)	0x00000000
0x310270B0	OSPI0_POLSTAT	OSPI0 Polling Flash Status Register	0x00000000
0x310270E0	OSPI0_OE_LWR	OSPI0 Opcode Extension Register (Lower)	0x13EDFA00
0x310270E4	OSPI0_OE_UP	OSPI0 Opcode Extension Register (Upper)	0x06F90000
0x310270FC	OSPI0_MODID	OSPI0 Module ID Register	0x04000300

Table A-71: ADSP-2156x OTPC0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31011004	OTPC0_STAT	OTPC0 OTP Status Register	0x00000000
0x3101102C	OTPC0_SECU_STATE	OTPC0 OTP Security State Register	0x00000001

Table A-72: ADSP-2156x PADS0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31004404	PADS0_PCFG0	PADS0 Peripheral PAD Configuration0 Register	0x00010000
0x3100440C	PADS0_PORTA0_DS	PADS0 PORTA 0 to 7 pins DS control	0x00249249
0x31004410	PADS0_PORTA1_DS	PADS0 PORTA 8 - 15 pins DS control	0x00249249
0x31004414	PADS0_PORTB0_DS	PADS0 PORTB 0 to 7 pins DS control	0x00249249
0x31004418	PADS0_PORTB1_DS	PADS0 PORTB 8 - 15 pins DS control	0x00249249

Table A-72: ADSP-2156x PADS0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3100441C	PADS0_PORTC0_DS	PADS0 PORTC 0 to 7 pins DS control	0x00249249
0x31004424	PADS0_NONPORTS_DS	PADS0 Non-GPIO Drive Strength Register	0x0001244A
0x31004434	PADS0_DAI0_0_DS	PADS0 DAI0 1 to 10 pins DS control	0x09249249
0x31004438	PADS0_DAI0_1_DS	PADS0 DAI0 11 to 20 pins DS control	0x09000009
0x31004440	PADS0_DAI1_0_DS	PADS0 DAI1 1 to 10 pins DS control	0x09249249
0x31004444	PADS0_DAI1_1_DS	PADS0 DAI1 11 to 20 pins DS control	0x09000009
0x31004460	PADS0_DAI0_IE	PADS0 DAI0 Port Input Enable Control Register	0x00000000
0x31004464	PADS0_DAI1_IE	PADS0 DAI1 Port Input Enable Control Register	0x00000000
0x31004480	PADS0_PORT[n]_PUE	PADS0 PORTx Pull-Up Enable	0x00000000
0x31004484	PADS0_PORT[n]_PUE	PADS0 PORTx Pull-Up Enable	0x00000000
0x31004488	PADS0_PORT[n]_PUE	PADS0 PORTx Pull-Up Enable	0x00000000
0x31004498	PADS0_DAI[n]_PUE	PADS0 DAIx Pull-Up Enable	0x00000000
0x3100449C	PADS0_DAI[n]_PUE	PADS0 DAIx Pull-Up Enable	0x00000000
0x310044C0	PADS0_PORT[n]_PDE	PADS0 PORTx Pull-Down Enable	0x00000000
0x310044C4	PADS0_PORT[n]_PDE	PADS0 PORTx Pull-Down Enable	0x00000000
0x310044C8	PADS0_PORT[n]_PDE	PADS0 PORTx Pull-Down Enable	0x00000000
0x310044D8	PADS0_DAI[n]_PDE	PADS0 DAIx Pull-Down Enable	0x00000000
0x310044DC	PADS0_DAI[n]_PDE	PADS0 DAIx Pull-Down Enable	0x00000000

Table A-73: ADSP-2156x PCG0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C9300	PCG0_CTLA0	PCG0 Precision Clock A Control 0 Register	0x00000000
0x310C9304	PCG0_CTLA1	PCG0 Precision Clock A Control 1 Register	0x00000000
0x310C9308	PCG0_CTLB0	PCG0 Precision Clock B Control 0 Register	0x00000000
0x310C930C	PCG0_CTLB1	PCG0 Precision Clock B Control 1 Register	0x00000000
0x310C9310	PCG0_PW1	PCG0 Precision Clock Pulse Width Control 1 Register	0x00000000
0x310C9314	PCG0_SYNC1	PCG0 Precision Clock Frame Sync Synchronization 1 Register	0x00000000
0x310CA300	PCG0_CTLC0	PCG0 Precision Clock C Control 0 Register	0x00000000
0x310CA304	PCG0_CTLC1	PCG0 Precision Clock C Control 1 Register	0x00000000
0x310CA308	PCG0_CTLD0	PCG0 Precision Clock D Control 0 Register	0x00000000

Table A-73: ADSP-2156x PCG0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CA30C	PCG0_CTLD1	PCG0 Precision Clock D Control 1 Register	0x00000000
0x310CA310	PCG0_PW2	PCG0 Precision Clock Pulse Width Control 2 Register	0x00000000
0x310CA314	PCG0_SYNC2	PCG0 Precision Clock Frame Sync Synchronization 2 Register	0x00000000

Table A-74: ADSP-2156x PINT0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31005000	PINT0_MSK_SET	PINT0 PINT Mask Set Register	0x00000000
0x31005004	PINT0_MSK_CLR	PINT0 PINT Mask Clear Register	0x00000000
0x31005008	PINT0_REQ	PINT0 PINT Request Register	0x00000000
0x3100500C	PINT0_ASSIGN	PINT0 PINT Assign Register	0x00000101
0x31005010	PINT0_EDGE_SET	PINT0 PINT Edge Set Register	0x00000000
0x31005014	PINT0_EDGE_CLR	PINT0 PINT Edge Clear Register	0x00000000
0x31005018	PINT0_INV_SET	PINT0 PINT Invert Set Register	0x00000000
0x3100501C	PINT0_INV_CLR	PINT0 PINT Invert Clear Register	0x00000000
0x31005020	PINT0_PINSTATE	PINT0 PINT Pin State Register	0x00000000
0x31005024	PINT0_LATCH	PINT0 PINT Latch Register	0x00000000

Table A-75: ADSP-2156x PINT1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31005100	PINT1_MSK_SET	PINT1 PINT Mask Set Register	0x00000000
0x31005104	PINT1_MSK_CLR	PINT1 PINT Mask Clear Register	0x00000000
0x31005108	PINT1_REQ	PINT1 PINT Request Register	0x00000000
0x3100510C	PINT1_ASSIGN	PINT1 PINT Assign Register	0x00000101
0x31005110	PINT1_EDGE_SET	PINT1 PINT Edge Set Register	0x00000000
0x31005114	PINT1_EDGE_CLR	PINT1 PINT Edge Clear Register	0x00000000
0x31005118	PINT1_INV_SET	PINT1 PINT Invert Set Register	0x00000000
0x3100511C	PINT1_INV_CLR	PINT1 PINT Invert Clear Register	0x00000000
0x31005120	PINT1_PINSTATE	PINT1 PINT Pin State Register	0x00000000
0x31005124	PINT1_LATCH	PINT1 PINT Latch Register	0x00000000

Table A-76: ADSP-2156x PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005200	PINT2_MSK_SET	PINT2 PINT Mask Set Register	0x00000000
0x31005204	PINT2_MSK_CLR	PINT2 PINT Mask Clear Register	0x00000000
0x31005208	PINT2_REQ	PINT2 PINT Request Register	0x00000000
0x3100520C	PINT2_ASSIGN	PINT2 PINT Assign Register	0x00000101
0x31005210	PINT2_EDGE_SET	PINT2 PINT Edge Set Register	0x00000000
0x31005214	PINT2_EDGE_CLR	PINT2 PINT Edge Clear Register	0x00000000
0x31005218	PINT2_INV_SET	PINT2 PINT Invert Set Register	0x00000000
0x3100521C	PINT2_INV_CLR	PINT2 PINT Invert Clear Register	0x00000000
0x31005220	PINT2_PINSTATE	PINT2 PINT Pin State Register	0x00000000
0x31005224	PINT2_LATCH	PINT2 PINT Latch Register	0x00000000

Table A-77: ADSP-2156x PKA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D4000	PKA0_APTR	PKA0 PKA Vector_A Address	0x00000000
0x310D4004	PKA0_BPTR	PKA0 PKA Vector_B Address	0x00000000
0x310D4008	PKA0_CPTR	PKA0 PKA Vector_C Address	0x00000000
0x310D400C	PKA0_DPTR	PKA0 PKA Vector_D Address	0x00000000
0x310D4010	PKA0_ALEN	PKA0 PKA Vector_A Length	0x00000000
0x310D4014	PKA0_BLEN	PKA0 PKA Vector_B Length	0x00000000
0x310D4018	PKA0_SHIFT	PKA0 PKA Bit Shift Value	0x00000000
0x310D401C	PKA0_FUNC	PKA0 PKA Function	0x00000000
0x310D4020	PKA0_COMPARE	PKA0 PKA Compare Result	0x00000001
0x310D4024	PKA0_RESULTMSW	PKA0 PKA Most-Significant-Word of Result Vector	0x00008000
0x310D4028	PKA0_DIVMSW	PKA0 PKA Most-Significant-Word of Divide Remainder	0x00008000
0x310D6000	PKA0_RAM	PKA0 Start of PKA RAM space	0x00000000

Table A-78: ADSP-2156x PKIC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D8000	PKIC0_POL_CTL	PKIC0 Polarity Control Register	0x00000000
0x310D8004	PKIC0_TYPE_CTL	PKIC0 Type Control Register	0x00000000

Table A-78: ADSP-2156x PKIC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310D8008	PKIC0_EN_CTL	PKIC0 Enable Control Register	0x00000000
0x310D800C	PKIC0_RAW_STAT	PKIC0 Raw Status Register	0x00000000
0x310D800C	PKIC0_EN_SET	PKIC0 Enable Set Register	0x00000000
0x310D8010	PKIC0_ACK	PKIC0 Acknowledge Register	0x00000000
0x310D8010	PKIC0_EN_STAT	PKIC0 Enabled Status Register	0x00000000
0x310D8014	PKIC0_EN_CLR	PKIC0 Enable Clear Register	0x00000000

Table A-79: ADSP-2156x PKTE0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CD000	PKTE0_CTL_STAT	PKTE0 Packet Engine Control Register	0x00000002
0x310CD004	PKTE0_SRC_ADDR	PKTE0 Packet Engine Source Address	0x00000000
0x310CD008	PKTE0_DEST_ADDR	PKTE0 Packet Engine Destination Address	0x00000000
0x310CD00C	PKTE0_SA_ADDR	PKTE0 Packet Engine SA Address	0x00000000
0x310CD010	PKTE0_STATE_ADDR	PKTE0 Packet Engine State Record Address	0x00000000
0x310CD014	PKTE0_ARC4STATE_ADDR	PKTE0 Packet Engine ARC4 State Record Address	0x00000000
0x310CD018	PKTE0_USERID	PKTE0 Packet Engine User ID	0x00000000
0x310CD01C	PKTE0_LEN	PKTE0 Packet Engine Length Register	0x00800000
0x310CD080	PKTE0_CDRBASE_ADDR	PKTE0 Packet Engine Command Descriptor Ring Base Address	0x00000000
0x310CD084	PKTE0_RDRBASE_ADDR	PKTE0 Packet Engine Result Descriptor Ring Base Address	0x00000000
0x310CD088	PKTE0_RING_CFG	PKTE0 Packet Engine Ring Configuration	0x00000000
0x310CD08C	PKTE0_RING_THRESH	PKTE0 Packet Engine Ring Threshold Registers	0x00000000
0x310CD090	PKTE0_CDSC_INCR	PKTE0 Packet Engine Command Descriptor Count Increment Register	0x00000000
0x310CD090	PKTE0_CDSC_CNT	PKTE0 Packet Engine Command Descriptor Count Register	0x00000000
0x310CD094	PKTE0_RDSC_CNT	PKTE0 Packet Engine Result Descriptor Count Registers	0x00000000
0x310CD094	PKTE0_RDSC_DECR	PKTE0 Packet Engine Result Descriptor Count Decrement Registers	0x00000000
0x310CD098	PKTE0_RING_PTR	PKTE0 Packet Engine Ring Pointer Status	0x00000000

Table A-79: ADSP-2156x PKTE0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CD09C	PKTE0_RING_STAT	PKTE0 Packet Engine Ring Status	0x00000000
0x310CD100	PKTE0_CFG	PKTE0 Packet Engine Configuration Register	0x00000000
0x310CD104	PKTE0_STAT	PKTE0 Packet Engine Status Register	0x00040402
0x310CD10C	PKTE0_BUF_THRESH	PKTE0 Packet Engine Buffer Threshold Register	0x00800080
0x310CD110	PKTE0_INBUF_INCR	PKTE0 Packet Engine Input Buffer Count Increment Register	0x00000000
0x310CD110	PKTE0_INBUF_CNT	PKTE0 Packet Engine Input Buffer Count Register	0x00000000
0x310CD114	PKTE0_OUTBUF_CNT	PKTE0 Packet Engine Output Buffer Count Register	0x00000000
0x310CD114	PKTE0_OUTBUF_DECR	PKTE0 Packet Engine Output Buffer Count Decrement Register	0x00000000
0x310CD118	PKTE0_BUF_PTR	PKTE0 Packet Engine Buffer Pointer Register	0x00000000
0x310CD120	PKTE0_DMA_CFG	PKTE0 Packet Engine DMA Configuration Register	0x00180006
0x310CD1D0	PKTE0_ENDIAN_CFG	PKTE0 Packet Engine Endian Configuration Register	0x00E400E4
0x310CD1E0	PKTE0_HLT_CTL	PKTE0 Packet Engine Halt Control Register	0x00000000
0x310CD1E0	PKTE0_HLT_STAT	PKTE0 Packet Engine Halt Status Register	0x00000000
0x310CD1E4	PKTE0_CONT	PKTE0 PKTE Continue Register	0x00000000
0x310CD1E8	PKTE0_CLK_CTL	PKTE0 PE Clock Control Register	0x0000001F
0x310CD200	PKTE0_IUMSK_STAT	PKTE0 Interrupt Unmasked Status Register	0x00000000
0x310CD204	PKTE0_IMSK_STAT	PKTE0 Interrupt Masked Status Register	0x00000000
0x310CD204	PKTE0_INT_CLR	PKTE0 Interrupt Clear Register	0x00000000
0x310CD208	PKTE0_INT_EN	PKTE0 Interrupt Enable Register	0x00000000
0x310CD20C	PKTE0_INT_CFG	PKTE0 Interrupt Configuration Register	0x00000000
0x310CD210	PKTE0_IMSK_EN	PKTE0 Interrupt Mask Enable Register	0x00000000
0x310CD214	PKTE0_IMSK_DIS	PKTE0 Interrupt Mask Disable Register	0x00000000
0x310CD400	PKTE0_SA_CMD0	PKTE0 SA Command 0	0x00000000
0x310CD404	PKTE0_SA_CMD1	PKTE0 SA Command 1	0x00000000
0x310CD408	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD40C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD410	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD414	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD418	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000

Table A-79: ADSP-2156x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD41C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD420	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD424	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD428	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD42C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD430	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD434	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD438	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD43C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD440	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD444	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD448	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD44C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD450	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD454	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD458	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD45C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD460	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD464	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD468	PKTE0_SA_SPI	PKTE0 SA SPI Register	0x00000000
0x310CD46C	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000
0x310CD470	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000
0x310CD474	PKTE0_SA_SEQ- NUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x310CD478	PKTE0_SA_SEQ- NUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x310CD47C	PKTE0_SA_ARC4IJPTR	PKTE0 ARC4 i and j Pointer Register	0x00000000
0x310CD47C	PKTE0_SA_RDY	PKTE0 SA Ready Indicator	0x00000000
0x310CD47C	PKTE0_SA_NONCE	PKTE0 SA Initialization Vector Register	0x00000000
0x310CD500	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD504	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000

Table A-79: ADSP-2156x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD508	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD50C	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD510	PKTE0_STATE_BYTE_COUNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x310CD514	PKTE0_STATE_BYTE_COUNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x310CD518	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD51C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD520	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD524	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD528	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD52C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD530	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD534	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD700	PKTE0_ARC4STATE_BUF	PKTE0 Starting Entry of 256-byte ARC4 State Buffer	0x00000000
0x310CD800	PKTE0_DATAIO_BUF	PKTE0 Starting Entry of 256-byte Data Input/Output Buffer	0x00000000

Table A-80: ADSP-2156x PORTA MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004000	PORTA_FER	PORTA Port x Function Enable Register	0x00000000
0x31004004	PORTA_FER_SET	PORTA Port x Function Enable Set Register	0x00000000
0x31004008	PORTA_FER_CLR	PORTA Port x Function Enable Clear Register	0x00000000
0x3100400C	PORTA_DATA	PORTA Port x GPIO Data Register	0x00000000
0x31004010	PORTA_DATA_SET	PORTA Port x GPIO Data Set Register	0x00000000
0x31004014	PORTA_DATA_CLR	PORTA Port x GPIO Data Clear Register	0x00000000
0x31004018	PORTA_DIR	PORTA Port x GPIO Direction Register	0x00000000
0x3100401C	PORTA_DIR_SET	PORTA Port x GPIO Direction Set Register	0x00000000
0x31004020	PORTA_DIR_CLR	PORTA Port x GPIO Direction Clear Register	0x00000000
0x31004024	PORTA_INEN	PORTA Port x GPIO Input Enable Register	0x00000000
0x31004028	PORTA_INEN_SET	PORTA Port x GPIO Input Enable Set Register	0x00000000

Table A-80: ADSP-2156x PORTA MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100402C	PORTA_INEN_CLR	PORTA Port x GPIO Input Enable Clear Register	0x00000000
0x31004030	PORTA_MUX	PORTA Port x Multiplexer Control Register	0x00000000
0x31004034	PORTA_DATA_TGL	PORTA Port x GPIO Output Toggle Register	0x00000000
0x31004038	PORTA_POL	PORTA Port x GPIO Polarity Invert Register	0x00000000
0x3100403C	PORTA_POL_SET	PORTA Port x GPIO Polarity Invert Set Register	0x00000000
0x31004040	PORTA_POL_CLR	PORTA Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004044	PORTA_LOCK	PORTA Port x GPIO Lock Register	0x00000000
0x31004048	PORTA_TRIG_TGL	PORTA Port x GPIO Trigger Toggle Register	0x00000000

Table A-81: ADSP-2156x PORTB MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004080	PORTB_FER	PORTB Port x Function Enable Register	0x00000000
0x31004084	PORTB_FER_SET	PORTB Port x Function Enable Set Register	0x00000000
0x31004088	PORTB_FER_CLR	PORTB Port x Function Enable Clear Register	0x00000000
0x3100408C	PORTB_DATA	PORTB Port x GPIO Data Register	0x00000000
0x31004090	PORTB_DATA_SET	PORTB Port x GPIO Data Set Register	0x00000000
0x31004094	PORTB_DATA_CLR	PORTB Port x GPIO Data Clear Register	0x00000000
0x31004098	PORTB_DIR	PORTB Port x GPIO Direction Register	0x00000000
0x3100409C	PORTB_DIR_SET	PORTB Port x GPIO Direction Set Register	0x00000000
0x310040A0	PORTB_DIR_CLR	PORTB Port x GPIO Direction Clear Register	0x00000000
0x310040A4	PORTB_INEN	PORTB Port x GPIO Input Enable Register	0x00000000
0x310040A8	PORTB_INEN_SET	PORTB Port x GPIO Input Enable Set Register	0x00000000
0x310040AC	PORTB_INEN_CLR	PORTB Port x GPIO Input Enable Clear Register	0x00000000
0x310040B0	PORTB_MUX	PORTB Port x Multiplexer Control Register	0x00000000
0x310040B4	PORTB_DATA_TGL	PORTB Port x GPIO Output Toggle Register	0x00000000
0x310040B8	PORTB_POL	PORTB Port x GPIO Polarity Invert Register	0x00000000
0x310040BC	PORTB_POL_SET	PORTB Port x GPIO Polarity Invert Set Register	0x00000000
0x310040C0	PORTB_POL_CLR	PORTB Port x GPIO Polarity Invert Clear Register	0x00000000
0x310040C4	PORTB_LOCK	PORTB Port x GPIO Lock Register	0x00000000
0x310040C8	PORTB_TRIG_TGL	PORTB Port x GPIO Trigger Toggle Register	0x00000000

Table A-82: ADSP-2156x PORTC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004100	PORTC_FER	PORTC Port x Function Enable Register	0x00000000
0x31004104	PORTC_FER_SET	PORTC Port x Function Enable Set Register	0x00000000
0x31004108	PORTC_FER_CLR	PORTC Port x Function Enable Clear Register	0x00000000
0x3100410C	PORTC_DATA	PORTC Port x GPIO Data Register	0x00000000
0x31004110	PORTC_DATA_SET	PORTC Port x GPIO Data Set Register	0x00000000
0x31004114	PORTC_DATA_CLR	PORTC Port x GPIO Data Clear Register	0x00000000
0x31004118	PORTC_DIR	PORTC Port x GPIO Direction Register	0x00000000
0x3100411C	PORTC_DIR_SET	PORTC Port x GPIO Direction Set Register	0x00000000
0x31004120	PORTC_DIR_CLR	PORTC Port x GPIO Direction Clear Register	0x00000000
0x31004124	PORTC_INEN	PORTC Port x GPIO Input Enable Register	0x00000000
0x31004128	PORTC_INEN_SET	PORTC Port x GPIO Input Enable Set Register	0x00000000
0x3100412C	PORTC_INEN_CLR	PORTC Port x GPIO Input Enable Clear Register	0x00000000
0x31004130	PORTC_MUX	PORTC Port x Multiplexer Control Register	0x00000000
0x31004134	PORTC_DATA_TGL	PORTC Port x GPIO Output Toggle Register	0x00000000
0x31004138	PORTC_POL	PORTC Port x GPIO Polarity Invert Register	0x00000000
0x3100413C	PORTC_POL_SET	PORTC Port x GPIO Polarity Invert Set Register	0x00000000
0x31004140	PORTC_POL_CLR	PORTC Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004144	PORTC_LOCK	PORTC Port x GPIO Lock Register	0x00000000
0x31004148	PORTC_TRIG_TGL	PORTC Port x GPIO Trigger Toggle Register	0x00000000

Table A-83: ADSP-2156x RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C000	RCU0_CTL	RCU0 Control Register	0x00000700
0x3108C004	RCU0_STAT	RCU0 Status Register	0x00000021
0x3108C008	RCU0_CRCTL	RCU0 Core Reset Outputs Control Register	0x00000000
0x3108C00C	RCU0_CRSTAT	RCU0 Core Reset Outputs Status Register	0x00000000
0x3108C018	RCU0_SRRQSTAT	RCU0 System Reset Request Status Register	0x00000000
0x3108C01C	RCU0_SIDIS	RCU0 System Interface Disable Register	0x00000000
0x3108C020	RCU0_SISTAT	RCU0 System Interface Status Register	0x00000000
0x3108C024	RCU0_SVECT_LCK	RCU0 SVECT Lock Register	0x00000000

Table A-83: ADSP-2156x RCU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108C028	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x3108C02C	RCU0_SVECT0	RCU0 Software Vector Register 0	0x00500004
0x3108C06C	RCU0_MSG	RCU0 Message Register	0x00000000
0x3108C070	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x3108C074	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000

Table A-84: ADSP-2156x SCB0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30042100	SCB0_SP0A_READ_QOS	SCB0 SP0A Read Quality of Service Register	0x0000000C
0x30042104	SCB0_SP0A_WRITE_QOS	SCB0 SP0A Write Quality of Service Register	0x0000000C
0x30043100	SCB0_SP0B_READ_QOS	SCB0 SP0B Read Quality of Service Register	0x0000000C
0x30043104	SCB0_SP0B_WRITE_QOS	SCB0 SP0B Write Quality of Service Register	0x0000000C
0x30044100	SCB0_SP1A_READ_QOS	SCB0 SP1A Read Quality of Service Register	0x0000000C
0x30044104	SCB0_SP1A_WRITE_QOS	SCB0 SP1A Write Quality of Service Register	0x0000000C
0x30045100	SCB0_SP1B_READ_QOS	SCB0 SP1B Read Quality of Service Register	0x0000000C
0x30045104	SCB0_SP1B_WRITE_QOS	SCB0 SP1B Write Quality of Service Register	0x0000000C
0x30046100	SCB0_SP2A_READ_QOS	SCB0 SP2A Read Quality of Service Register	0x0000000C
0x30046104	SCB0_SP2A_WRITE_QOS	SCB0 SP2A Write Quality of Service Register	0x0000000C
0x30047100	SCB0_SP2B_READ_QOS	SCB0 SP2B Read Quality of Service Register	0x0000000C
0x30047104	SCB0_SP2B_WRITE_QOS	SCB0 SP2B Write Quality of Service Register	0x0000000C
0x30048100	SCB0_SP3A_READ_QOS	SCB0 SP3A Read Quality of Service Register	0x0000000C
0x30048104	SCB0_SP3A_WRITE_QOS	SCB0 SP3A Write Quality of Service Register	0x0000000C
0x30049100	SCB0_SP3B_READ_QOS	SCB0 SP3B Read Quality of Service Register	0x0000000C
0x30049104	SCB0_SP3B_WRITE_QOS	SCB0 SP3B Write Quality of Service Register	0x0000000C
0x3004A100	SCB0_CRC0_CH0_READ_QOS	SCB0 CRC0 Channel 0 Read Quality of Service Register	0x00000001
0x3004A104	SCB0_CRC0_CH0_WRITE_QOS	SCB0 CRC0 Channel 0 Write Quality of Service Register	0x00000001
0x3004B100	SCB0_CRC0_CH1_READ_QOS	SCB0 CRC0 Channel 1 Read Quality of Service Register	0x00000001
0x3004B104	SCB0_CRC0_CH1_WRITE_QOS	SCB0 CRC0 Channel 1 Write Quality of Service Register	0x00000001

Table A-84: ADSP-2156x SCB0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3004C100	SCB0_CRC1_CH0_READ_QOS	SCB0 CRC1 Channel 0 Read Quality of Service Register	0x00000001
0x3004C104	SCB0_CRC1_CH0_WRITE_QOS	SCB0 CRC1 Channel 0 Write Quality of Service Register	0x00000001
0x3004D100	SCB0_CRC1_CH1_READ_QOS	SCB0 CRC1 Channel 1 Read Quality of Service Register	0x00000001
0x3004D104	SCB0_CRC1_CH1_WRITE_QOS	SCB0 CRC1 Channel 1 Write Quality of Service Register	0x00000001
0x3004E100	SCB0_UART2_RX_READ_QOS	SCB0 UART2 RX Read Quality of Service Register	0x0000000A
0x3004E104	SCB0_UART2_RX_WRITE_QOS	SCB0 UART2 RX Write Quality of Service Register	0x0000000A
0x3004F100	SCB0_SH0_DPORT_READ_QOS	SCB0 SH0 DPORT Read Quality of Service Register	0x00000007
0x3004F104	SCB0_SH0_DPORT_WRITE_QOS	SCB0 SH0 DPORT Write Quality of Service Register	0x00000007
0x30050100	SCB0_SH0_IPORT_READ_QOS	SCB0 SH0 IPORT Read Quality of Service Register	0x00000007
0x30050104	SCB0_SH0_IPORT_WRITE_QOS	SCB0 SH0 IPORT Write Quality of Service Register	0x00000007
0x30051100	SCB0_SP4A_READ_QOS	SCB0 SP4A Read Quality of Service Register	0x0000000C
0x30051104	SCB0_SP4A_WRITE_QOS	SCB0 SP4A Write Quality of Service Register	0x0000000C
0x30052100	SCB0_SP4B_READ_QOS	SCB0 SP4B Read Quality of Service Register	0x0000000C
0x30052104	SCB0_SP4B_WRITE_QOS	SCB0 SP4B Write Quality of Service Register	0x0000000C
0x30055100	SCB0_HSMDMA_CH0_READ_QOS	SCB0 HSMDMA Channel 0 Read Quality of Service Register	0x00000001
0x30055104	SCB0_HSMDMA_CH0_WRITE_QOS	SCB0 HSMDMA Channel 0 Write Quality of Service Register	0x00000001
0x30056100	SCB0_SP5A_READ_QOS	SCB0 SP5A Read Quality of Service Register	0x0000000C
0x30056104	SCB0_SP5A_WRITE_QOS	SCB0 SP5A Write Quality of Service Register	0x0000000C
0x30057100	SCB0_MLB_READ_QOS	SCB0 MLB Read Quality of Service Register	0x0000000C
0x30057104	SCB0_MLB_WRITE_QOS	SCB0 MLB Write Quality of Service Register	0x0000000C
0x30058100	SCB0_UART0_TX_READ_QOS	SCB0 UART0 TX Read Quality of Service Register	0x0000000A

Table A-84: ADSP-2156x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30058104	SCB0_UART0_TX_WRITE_QOS	SCB0 UART0 TX Write Quality of Service Register	0x0000000A
0x30059100	SCB0_UART0_RX_READ_QOS	SCB0 UART0 RX Read Quality of Service Register	0x0000000A
0x30059104	SCB0_UART0_RX_WRITE_QOS	SCB0 UART0 RX Write Quality of Service Register	0x0000000A
0x3005A100	SCB0_SP5B_READ_QOS	SCB0 SP5B Read Quality of Service Register	0x0000000C
0x3005A104	SCB0_SP5B_WRITE_QOS	SCB0 SP5B Write Quality of Service Register	0x0000000C
0x3005B100	SCB0_HSMDMA_CH1_READ_QOS	SCB0 HSMDMA Channel 1 Read Quality of Service Register	0x00000001
0x3005B104	SCB0_HSMDMA_CH1_WRITE_QOS	SCB0 HSMDMA Channel 1 Write Quality of Service Register	0x00000001
0x3005C100	SCB0_UART2_TX_READ_QOS	SCB0 UART2 TX Read Quality of Service Register	0x0000000A
0x3005C104	SCB0_UART2_TX_WRITE_QOS	SCB0 UART2 TX Write Quality of Service Register	0x0000000A
0x3005D100	SCB0_SPI0TX_READ_QOS	SCB0 SPI0 TX Read Quality of Service Register	0x0000000C
0x3005D104	SCB0_SPI0TX_WRITE_QOS	SCB0 SPI0 TX Write Quality of Service Register	0x0000000C
0x3005E100	SCB0_SPI0RX_READ_QOS	SCB0 SPI0 RX Read Quality of Service Register	0x0000000C
0x3005E104	SCB0_SPI0RX_WRITE_QOS	SCB0 SPI0 RX Write Quality of Service Register	0x0000000C
0x3005F100	SCB0_SPI1TX_READ_QOS	SCB0 SPI1 TX Read Quality of Service Register	0x0000000C
0x3005F104	SCB0_SPI1TX_WRITE_QOS	SCB0 SPI1 TX Write Quality of Service Register	0x0000000C
0x30060100	SCB0_SPI1RX_READ_QOS	SCB0 SPI1 RX Read Quality of Service Register	0x0000000C
0x30060104	SCB0_SPI1RX_WRITE_QOS	SCB0 SPI1 RX Write Quality of Service Register	0x0000000C
0x30061100	SCB0_SPI2TX_READ_QOS	SCB0 SPI2 TX Read Quality of Service Register	0x0000000C
0x30061104	SCB0_SPI2TX_WRITE_QOS	SCB0 SPI2 TX Write Quality of Service Register	0x0000000C
0x30062100	SCB0_SPI2RX_READ_QOS	SCB0 SPI2 RX Read Quality of Service Register	0x0000000C
0x30062104	SCB0_SPI2RX_WRITE_QOS	SCB0 SPI2 RX Write Quality of Service Register	0x0000000C
0x30063100	SCB0_SP6A_READ_QOS	SCB0 SP6A Read Quality of Service Register	0x0000000C

Table A-84: ADSP-2156x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30063104	SCB0_SP6A_WRITE_QOS	SCB0 SP6A Write Quality of Service Register	0x0000000C
0x30064100	SCB0_SP6B_READ_QOS	SCB0 SP6B Read Quality of Service Register	0x0000000C
0x30064104	SCB0_SP6B_WRITE_QOS	SCB0 SP6B Write Quality of Service Register	0x0000000C
0x30065100	SCB0_LP0_READ_QOS	SCB0 LP0 Read Quality of Service Register	0x00000001
0x30065104	SCB0_LP0_WRITE_QOS	SCB0 LP0 Write Quality of Service Register	0x00000001
0x30066100	SCB0_SP7A_READ_QOS	SCB0 SP7A Read Quality of Service Register	0x0000000C
0x30066104	SCB0_SP7A_WRITE_QOS	SCB0 SP7A Write Quality of Service Register	0x0000000C
0x30067100	SCB0_SP7B_READ_QOS	SCB0 SP7B Read Quality of Service Register	0x0000000C
0x30067104	SCB0_SP7B_WRITE_QOS	SCB0 SP7B Write Quality of Service Register	0x0000000C
0x30068100	SCB0_SH0_MMR_IB_READ_QOS	SCB0 SH0 MMR Read Quality of Service Register	0x00000007
0x30068104	SCB0_SH0_MMR_IB_WRITE_QOS	SCB0 SH0 MMR Write Quality of Service Register	0x00000007
0x3006A100	SCB0_UART1_TX_READ_QOS	SCB0 UART1 TX Read Quality of Service Register	0x0000000A
0x3006A104	SCB0_UART1_TX_WRITE_QOS	SCB0 UART1 TX Write Quality of Service Register	0x0000000A
0x3006B100	SCB0_UART1_RX_READ_QOS	SCB0 UART1 RX Read Quality of Service Register	0x0000000A
0x3006B104	SCB0_UART1_RX_WRITE_QOS	SCB0 UART1 RX Write Quality of Service Register	0x0000000A
0x3006D100	SCB0_LP1_READ_QOS	SCB0 LP1 Read Quality of Service Register	0x00000001
0x3006D104	SCB0_LP1_WRITE_QOS	SCB0 LP1 Write Quality of Service Register	0x00000001
0x30071100	SCB0_CRYPT-TO_READ_QOS	SCB0 Crypto Read Quality of Service Register	0x00000001
0x30071104	SCB0_CRYPT-TO_WRITE_QOS	SCB0 CRYPTO Write Quality of Service Register	0x00000001
0x30074100	SCB0_ACC_CH0_READ_QOS	SCB0 Accelerator Channel 0 Read Quality of Service Register	0x00000001
0x30074104	SCB0_ACC_CH0_WRITE_QOS	SCB0 Accelerator Channel 0 Write Quality of Service Register	0x00000001
0x30075100	SCB0_ACC_CH1_READ_QOS	SCB0 Accelerator Channel 1 Read Quality of Service Register	0x00000001
0x30075104	SCB0_ACC_CH1_WRITE_QOS	SCB0 Accelerator Channel 1 Write Quality of Service Register	0x00000001

Table A-84: ADSP-2156x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30078100	SCB0_DLDMA0_CH0_READ_QOS	SCB0 DLDMA0 Channel 0 Read Quality of Service Register	0x00000001
0x30078104	SCB0_DLDMA0_CH0_WRITE_QOS	SCB0 DLDMA0 Channel 0 Write Quality of Service Register	0x00000001
0x30079100	SCB0_DLDMA0_CH1_READ_QOS	SCB0 DLDMA0 Channel 1 Read Quality of Service Register	0x00000001
0x30079104	SCB0_DLDMA0_CH1_WRITE_QOS	SCB0 DLDMA0 Channel 1 Write Quality of Service Register	0x00000001
0x3007A100	SCB0_DLDMA1_CH0_READ_QOS	SCB0 DLDMA1 Channel 0 Read Quality of Service Register	0x00000001
0x3007A104	SCB0_DLDMA1_CH0_WRITE_QOS	SCB0 DLDMA1 Channel 0 Write Quality of Service Register	0x00000001
0x3007B100	SCB0_DLDMA1_CH1_READ_QOS	SCB0 DLDMA1 Channel 1 Read Quality of Service Register	0x00000001
0x3007B104	SCB0_DLDMA1_CH1_WRITE_QOS	SCB0 DLDMA1 Channel 1 Write Quality of Service Register	0x00000001
0x3007C100	SCB0_MSMDMA_CH0_READ_QOS	SCB0 MSMDMA Channel 0 Read Quality of Service Register	0x00000001
0x3007C104	SCB0_MSMDMA_CH0_WRITE_QOS	SCB0 MSMDMA Channel 0 Write Quality of Service Register	0x00000001
0x3007D100	SCB0_MSMDMA_CH1_READ_QOS	SCB0 MSMDMA Channel 1 Read Quality of Service Register	0x00000001
0x3007D104	SCB0_MSMDMA_CH1_WRITE_QOS	SCB0 MSMDMA Channel 1 Write Quality of Service Register	0x00000001
0x30088100	SCB0_DBG_READ_QOS	SCB0 DBG Read Quality of Service Register	0x00000003
0x30088104	SCB0_DBG_WRITE_QOS	SCB0 DBG Write Quality of Service Register	0x00000003
0x30089100	SCB0_ETR_READ_QOS	SCB0 ETR Read Quality of Service Register	0x00000003
0x30089104	SCB0_ETR_WRITE_QOS	SCB0 ETR Write Quality of Service Register	0x00000003

Table A-85: ADSP-2156x SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31089000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x31089004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x31089008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map- ped Address	Register Name	Description	Reset Value
0x3108900C	SEC0_END	SEC0 Global End Register	0x00000000
0x31089010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000
0x31089014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x31089018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x3108901C	SEC0_FEND	SEC0 Fault End Register	0x00000000
0x31089020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x31089024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x31089028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x3108902C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x31089030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x31089034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x31089400	SEC0_CCTL[n]	SEC0 SCI Control Register n	0x00000000
0x31089404	SEC0_CSTAT[n]	SEC0 SCI Status Register n	0x00000000
0x31089408	SEC0_CPND[n]	SEC0 Core Pending Register n	0x00000000
0x3108940C	SEC0_CACT[n]	SEC0 SCI Active Register n	0x00000000
0x31089410	SEC0_CPMSK[n]	SEC0 SCI Priority Mask Register n	0x000000FF
0x31089414	SEC0_CGMSK[n]	SEC0 SCI Group Mask Register n	0x00000000
0x31089418	SEC0_CPLVL[n]	SEC0 SCI Priority Level Register n	0x00000007
0x3108941C	SEC0_CSID[n]	SEC0 SCI Source ID Register n	0x00000000
0x31089800	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089804	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089808	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108980C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089810	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089814	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089818	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108981C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089820	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089824	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089828	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108982C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089830	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089834	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089838	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108983C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089840	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089844	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089848	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108984C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089850	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089854	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089858	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108985C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089860	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089864	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089868	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108986C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089870	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089874	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089878	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108987C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089880	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089884	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089888	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108988C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089890	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089894	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089898	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108989C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310898AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089900	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089904	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089908	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108990C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089910	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089914	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089918	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108991C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089920	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089924	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089928	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108992C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089930	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089934	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089938	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108993C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089940	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089944	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089948	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108994C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089950	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089954	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089958	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108995C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089960	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089964	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089968	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108996C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089970	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089974	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089978	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108997C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089980	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089984	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089988	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108998C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089990	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089994	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089998	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108999C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310899A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089A20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089A9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ABC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ACC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ADC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089B18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089B94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089C10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089C8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089D08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089D84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-85: ADSP-2156x SEC0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31089E00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-86: ADSP-2156x SCB4 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x30301FD0	SCB4_PERIPH_ID_4	SCB4 Periph Id 4	0x00000004
0x30301FD4	SCB4_PERIPH_ID_5	SCB4 Periph Id 5	0x00000000
0x30301FD8	SCB4_PERIPH_ID_6	SCB4 Periph Id 6	0x00000000
0x30301FDC	SCB4_PERIPH_ID_7	SCB4 Periph Id 7	0x00000000
0x30301FE0	SCB4_PERIPH_ID_0	SCB4 Periph Id 0	0x00000001
0x30301FE4	SCB4_PERIPH_ID_1	SCB4 Periph Id 1	0x000000B3
0x30301FE8	SCB4_PERIPH_ID_2	SCB4 Periph Id 2	0x0000007B
0x30301FEC	SCB4_PERIPH_ID_3	SCB4 Periph Id 3	0x00000000
0x30301FF0	SCB4_COMP_ID_0	SCB4 Comp Id 0	0x0000000D
0x30301FF4	SCB4_COMP_ID_1	SCB4 Comp Id 1	0x000000F0
0x30301FF8	SCB4_COMP_ID_2	SCB4 Comp Id 2	0x00000005
0x30301FFC	SCB4_COMP_ID_3	SCB4 Comp Id 3	0x000000B1
0x30343100	SCB4_SHARC_DPORT_READ_QOS	SCB4 Sharc Dport.read Qos	0x0000000F
0x30343104	SCB4_SHARC_DPORT_WRITE_QOS	SCB4 Sharc Dport.write Qos	0x0000000F
0x30343108	SCB4_SHARC_DPORT_FN_MOD	SCB4 Sharc Dport.fn Mod	0x00000000
0x30344100	SCB4_FABRIC_S2PORT_IB_READ_QOS	SCB4 Fabric S2port Ib.read Qos	0x0000000F
0x30344104	SCB4_FABRIC_S2PORT_IB_WRITE_QOS	SCB4 Fabric S2port Ib.write Qos	0x0000000F
0x30344108	SCB4_FABRIC_S2PORT_IB_FN_MOD	SCB4 Fabric S2port Ib.fn Mod	0x00000000
0x30345024	SCB4_FIR_CH0_IB_FN_MOD2	SCB4 Fir Ch0 Ib.fn Mod2	0x00000000

Table A-86: ADSP-2156x SCB4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30345100	SCB4_FIR_CH0_IB_READ_QOS	SCB4 Fir Ch0 Ib.read Qos	0x0000000F
0x30345104	SCB4_FIR_CH0_IB_WRITE_QOS	SCB4 Fir Ch0 Ib.write Qos	0x0000000F
0x30345108	SCB4_FIR_CH0_IB_FN_MOD	SCB4 Fir Ch0 Ib.fn Mod	0x00000000
0x30346024	SCB4_FIR_CH1_IB_FN_MOD2	SCB4 Fir Ch1 Ib.fn Mod2	0x00000000
0x30346100	SCB4_FIR_CH1_IB_READ_QOS	SCB4 Fir Ch1 Ib.read Qos	0x0000000F
0x30346104	SCB4_FIR_CH1_IB_WRITE_QOS	SCB4 Fir Ch1 Ib.write Qos	0x0000000F
0x30346108	SCB4_FIR_CH1_IB_FN_MOD	SCB4 Fir Ch1 Ib.fn Mod	0x00000000
0x30347024	SCB4_IIR_CH0_IB_FN_MOD2	SCB4 Iir Ch0 Ib.fn Mod2	0x00000000
0x30347100	SCB4_IIR_CH0_IB_READ_QOS	SCB4 Iir Ch0 Ib.read Qos	0x0000000F
0x30347104	SCB4_IIR_CH0_IB_WRITE_QOS	SCB4 Iir Ch0 Ib.write Qos	0x0000000F
0x30347108	SCB4_IIR_CH0_IB_FN_MOD	SCB4 Iir Ch0 Ib.fn Mod	0x00000000
0x30348024	SCB4_IIR_CH1_IB_FN_MOD2	SCB4 Iir Ch1 Ib.fn Mod2	0x00000000
0x30348100	SCB4_IIR_CH1_IB_READ_QOS	SCB4 Iir Ch1 Ib.read Qos	0x0000000F
0x30348104	SCB4_IIR_CH1_IB_WRITE_QOS	SCB4 Iir Ch1 Ib.write Qos	0x0000000F
0x30348108	SCB4_IIR_CH1_IB_FN_MOD	SCB4 Iir Ch1 Ib.fn Mod	0x00000000

Table A-87: ADSP-2156x SMPU11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A1000	SMPU11_CTL	SMPU11 SMPU Control Register	0x00000000
0x310A1004	SMPU11_STAT	SMPU11 SMPU Status Register	0x00000000

Table A-87: ADSP-2156x SMPU11 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A1008	SMPU11_IADDR	SMPU11 Interrupt Address Register	0x00000000
0x310A100C	SMPU11_IDTLS	SMPU11 Interrupt Details Register	0x00000000
0x310A1010	SMPU11_BADDR	SMPU11 Bus Error Address Register	0x00000000
0x310A1014	SMPU11_BDTLS	SMPU11 Bus Error Details Register	0x00000000
0x310A1020	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A1024	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A1028	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A102C	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A1030	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A1034	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1038	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A103C	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A1040	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A1044	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A1048	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A104C	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1050	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A1054	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A1058	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A105C	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A1060	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A1064	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1068	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A106C	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A1070	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A1074	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A1078	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A107C	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1080	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A1084	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A1088	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000

Table A-87: ADSP-2156x SMPU11 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A108C	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A1090	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A1094	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1098	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A109C	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A10A0	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A10A4	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A10A8	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A10AC	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A10B0	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A10B4	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A10B8	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A10BC	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A10C0	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A10C4	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A10C8	SMPU11_RCTL[n]	SMPU11 Region n Control Register	0x00000000
0x310A10CC	SMPU11_RADDR[n]	SMPU11 Region n Address Register	0x00000000
0x310A10D0	SMPU11_RIDA[n]	SMPU11 Region n ID A Register	0x00000000
0x310A10D4	SMPU11_RIDMSKA[n]	SMPU11 Region n ID Mask A Register	0x00000000
0x310A10D8	SMPU11_RIDB[n]	SMPU11 Region n ID B Register	0x00000000
0x310A10DC	SMPU11_RIDMSKB[n]	SMPU11 Region n ID Mask B Register	0x00000000
0x310A1220	SMPU11_REVID	SMPU11 SMPU Revision ID Register	0x00000010
0x310A1800	SMPU11_SECURECTL	SMPU11 SMPU Control Secure Accesses Register	0x00000000
0x310A1820	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A1824	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A1828	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A182C	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A1830	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A1834	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A1838	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000
0x310A183C	SMPU11_SECURERCTL[n]	SMPU11 Region n Control Secure Accesses Register	0x00000000

Table A-88: ADSP-2156x SMPU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31083000	SMPU2_CTL	SMPU2 SMPU Control Register	0x00000000
0x31083004	SMPU2_STAT	SMPU2 SMPU Status Register	0x00000000
0x31083008	SMPU2_IADDR	SMPU2 Interrupt Address Register	0x00000000
0x3108300C	SMPU2_IDTLS	SMPU2 Interrupt Details Register	0x00000000
0x31083010	SMPU2_BADDR	SMPU2 Bus Error Address Register	0x00000000
0x31083014	SMPU2_BDTLS	SMPU2 Bus Error Details Register	0x00000000
0x31083020	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x31083024	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083028	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108302C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083030	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083034	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083038	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x3108303C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083040	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x31083044	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083048	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x3108304C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083050	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x31083054	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083058	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108305C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083060	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083064	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083068	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x3108306C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083070	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x31083074	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083078	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x3108307C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083080	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000

Table A-88: ADSP-2156x SMPU2 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31083084	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083088	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108308C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083090	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083094	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083098	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x3108309C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830A0	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830A4	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830A8	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830AC	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x310830B0	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x310830B4	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830B8	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830BC	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830C0	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830C4	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x310830C8	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x310830CC	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830D0	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830D4	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830D8	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830DC	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083220	SMPU2_REVID	SMPU2 SMPU Revision ID Register	0x00000010
0x31083800	SMPU2_SECURECTL	SMPU2 SMPU Control Secure Accesses Register	0x00000000
0x31083820	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083824	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083828	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x3108382C	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083830	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083834	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000

Table A-88: ADSP-2156x SMPU2 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31083838	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x3108383C	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000

Table A-89: ADSP-2156x SMPU3 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31084000	SMPU3_CTL	SMPU3 SMPU Control Register	0x00000000
0x31084004	SMPU3_STAT	SMPU3 SMPU Status Register	0x00000000
0x31084008	SMPU3_IADDR	SMPU3 Interrupt Address Register	0x00000000
0x3108400C	SMPU3_IDTLS	SMPU3 Interrupt Details Register	0x00000000
0x31084010	SMPU3_BADDR	SMPU3 Bus Error Address Register	0x00000000
0x31084014	SMPU3_BDTLS	SMPU3 Bus Error Details Register	0x00000000
0x31084020	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084024	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084028	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108402C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084030	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x31084034	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084038	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x3108403C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084040	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x31084044	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084048	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x3108404C	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084050	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084054	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084058	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108405C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084060	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x31084064	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084068	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000

Table A-89: ADSP-2156x SMPU3 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108406C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084070	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x31084074	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084078	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x3108407C	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084080	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084084	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084088	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108408C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084090	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x31084094	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084098	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x3108409C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840A0	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840A4	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840A8	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840AC	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x310840B0	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x310840B4	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840B8	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840BC	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840C0	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840C4	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x310840C8	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x310840CC	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840D0	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840D4	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840D8	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840DC	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084220	SMPU3_REVID	SMPU3 SMPU Revision ID Register	0x00000010
0x31084800	SMPU3_SECURECTL	SMPU3 SMPU Control Secure Accesses Register	0x00000000

Table A-89: ADSP-2156x SMPU3 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31084820	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084824	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084828	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x3108482C	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084830	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084834	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084838	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x3108483C	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000

Table A-90: ADSP-2156x SMPU9 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A0000	SMPU9_CTL	SMPU9 SMPU Control Register	0x00000000
0x310A0004	SMPU9_STAT	SMPU9 SMPU Status Register	0x00000000
0x310A0008	SMPU9_IADDR	SMPU9 Interrupt Address Register	0x00000000
0x310A000C	SMPU9_IDTLS	SMPU9 Interrupt Details Register	0x00000000
0x310A0010	SMPU9_BADDR	SMPU9 Bus Error Address Register	0x00000000
0x310A0014	SMPU9_BDTLS	SMPU9 Bus Error Details Register	0x00000000
0x310A0020	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A0024	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0028	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A002C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0030	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0034	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0038	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A003C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0040	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A0044	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0048	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A004C	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0050	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000

Table A-90: ADSP-2156x SMPU9 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A0054	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0058	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A005C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0060	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0064	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0068	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A006C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0070	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A0074	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0078	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A007C	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0080	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A0084	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0088	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A008C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0090	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0094	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0098	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A009C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A00A0	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00A4	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00A8	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00AC	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A00B0	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A00B4	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A00B8	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00BC	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00C0	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00C4	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A00C8	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A00CC	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000

Table A-90: ADSP-2156x SMPU9 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A00D0	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00D4	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00D8	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00DC	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0220	SMPU9_REVID	SMPU9 SMPU Revision ID Register	0x00000010
0x310A0800	SMPU9_SECURECTL	SMPU9 SMPU Control Secure Accesses Register	0x00000000
0x310A0820	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0824	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0828	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A082C	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0830	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0834	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0838	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A083C	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000

Table A-91: ADSP-2156x SPDIF0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C9280	SPDIF0_TX_CTL	SPDIF0 Transmit Control Register	0x00000000
0x310C9284	SPDIF0_TX_STAT_A0	SPDIF0 Transmit Status A0 Register	0x00000000
0x310C9288	SPDIF0_TX_STAT_B0	SPDIF0 Transmit Status B0 Register	0x00000000
0x310C92A0	SPDIF0_RX_CTL	SPDIF0 Receive Control	0x00000018
0x310C92A4	SPDIF0_RX_STAT	SPDIF0 Receive Status Register	0x00000000
0x310C92A8	SPDIF0_RX_STAT0_A	SPDIF0 Receive Status A0 Register	0x00000000
0x310C92AC	SPDIF0_RX_STAT0_B	SPDIF0 Receive Status B0 Register	0x00000000
0x310C92B0	SPDIF0_RX_STAT1_A	SPDIF0 Receive Status A1 Register	0x00000000
0x310C92B4	SPDIF0_RX_STAT1_B	SPDIF0 Receive Status B1 Register	0x00000000
0x310C9350	SPDIF0_TX_STAT_A1	SPDIF0 Transmit Status A1 Register	0x00000000
0x310C9354	SPDIF0_TX_STAT_A2	SPDIF0 Transmit Status A2 Register	0x00000000
0x310C9358	SPDIF0_TX_STAT_A3	SPDIF0 Transmit Status A3 Register	0x00000000
0x310C935C	SPDIF0_TX_STAT_A4	SPDIF0 Transmit Status A4 Register	0x00000000

Table A-91: ADSP-2156x SPDIF0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310C9360	SPDIF0_TX_STAT_A5	SPDIF0 Transmit Status A5 Register	0x00000000
0x310C9368	SPDIF0_TX_STAT_B1	SPDIF0 Transmit Status B1 Register	0x00000000
0x310C936C	SPDIF0_TX_STAT_B2	SPDIF0 Transmit Status B2 Register	0x00000000
0x310C9370	SPDIF0_TX_STAT_B3	SPDIF0 Transmit Status B3 Register	0x00000000
0x310C9374	SPDIF0_TX_STAT_B4	SPDIF0 Transmit Status B4 Register	0x00000000
0x310C9378	SPDIF0_TX_STAT_B5	SPDIF0 Transmit Status B5 Register	0x00000000
0x310C9380	SPDIF0_TX_UBUFF_A0	SPDIF0 Transmit User Buffer A0 Register	0x00000000
0x310C9384	SPDIF0_TX_UBUFF_A1	SPDIF0 Transmit User Buffer A1 Register	0x00000000
0x310C9388	SPDIF0_TX_UBUFF_A2	SPDIF0 Transmit User Buffer A2 Register	0x00000000
0x310C938C	SPDIF0_TX_UBUFF_A3	SPDIF0 Transmit User Buffer A3 Register	0x00000000
0x310C9390	SPDIF0_TX_UBUFF_A4	SPDIF0 Transmit User Buffer A4 Register	0x00000000
0x310C9394	SPDIF0_TX_UBUFF_A5	SPDIF0 Transmit User Buffer A5 Register	0x00000000
0x310C93A0	SPDIF0_TX_UBUFF_B0	SPDIF0 Transmit User Buffer B0 Register	0x00000000
0x310C93A4	SPDIF0_TX_UBUFF_B1	SPDIF0 Transmit User Buffer B1 Register	0x00000000
0x310C93A8	SPDIF0_TX_UBUFF_B2	SPDIF0 Transmit User Buffer B2 Register	0x00000000
0x310C93AC	SPDIF0_TX_UBUFF_B3	SPDIF0 Transmit User Buffer B3 Register	0x00000000
0x310C93B0	SPDIF0_TX_UBUFF_B4	SPDIF0 Transmit User Buffer B4 Register	0x00000000
0x310C93B4	SPDIF0_TX_UBUFF_B5	SPDIF0 Transmit User Buffer B5 Register	0x00000000
0x310C93BC	SPDIF0_TX_USRUPDT	SPDIF0 User Bit Update Register	0x00000000

Table A-92: ADSP-2156x SPDIF1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310CA280	SPDIF1_TX_CTL	SPDIF1 Transmit Control Register	0x00000000
0x310CA284	SPDIF1_TX_STAT_A0	SPDIF1 Transmit Status A0 Register	0x00000000
0x310CA288	SPDIF1_TX_STAT_B0	SPDIF1 Transmit Status B0 Register	0x00000000
0x310CA2A0	SPDIF1_RX_CTL	SPDIF1 Receive Control	0x00000018
0x310CA2A4	SPDIF1_RX_STAT	SPDIF1 Receive Status Register	0x00000000
0x310CA2A8	SPDIF1_RX_STAT0_A	SPDIF1 Receive Status A0 Register	0x00000000
0x310CA2AC	SPDIF1_RX_STAT0_B	SPDIF1 Receive Status B0 Register	0x00000000
0x310CA2B0	SPDIF1_RX_STAT1_A	SPDIF1 Receive Status A1 Register	0x00000000

Table A-92: ADSP-2156x SPDIF1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CA2B4	SPDIF1_RX_STAT1_B	SPDIF1 Receive Status B1 Register	0x00000000
0x310CA350	SPDIF1_TX_STAT_A1	SPDIF1 Transmit Status A1 Register	0x00000000
0x310CA354	SPDIF1_TX_STAT_A2	SPDIF1 Transmit Status A2 Register	0x00000000
0x310CA358	SPDIF1_TX_STAT_A3	SPDIF1 Transmit Status A3 Register	0x00000000
0x310CA35C	SPDIF1_TX_STAT_A4	SPDIF1 Transmit Status A4 Register	0x00000000
0x310CA360	SPDIF1_TX_STAT_A5	SPDIF1 Transmit Status A5 Register	0x00000000
0x310CA368	SPDIF1_TX_STAT_B1	SPDIF1 Transmit Status B1 Register	0x00000000
0x310CA36C	SPDIF1_TX_STAT_B2	SPDIF1 Transmit Status B2 Register	0x00000000
0x310CA370	SPDIF1_TX_STAT_B3	SPDIF1 Transmit Status B3 Register	0x00000000
0x310CA374	SPDIF1_TX_STAT_B4	SPDIF1 Transmit Status B4 Register	0x00000000
0x310CA378	SPDIF1_TX_STAT_B5	SPDIF1 Transmit Status B5 Register	0x00000000
0x310CA380	SPDIF1_TX_UBUFF_A0	SPDIF1 Transmit User Buffer A0 Register	0x00000000
0x310CA384	SPDIF1_TX_UBUFF_A1	SPDIF1 Transmit User Buffer A1 Register	0x00000000
0x310CA388	SPDIF1_TX_UBUFF_A2	SPDIF1 Transmit User Buffer A2 Register	0x00000000
0x310CA38C	SPDIF1_TX_UBUFF_A3	SPDIF1 Transmit User Buffer A3 Register	0x00000000
0x310CA390	SPDIF1_TX_UBUFF_A4	SPDIF1 Transmit User Buffer A4 Register	0x00000000
0x310CA394	SPDIF1_TX_UBUFF_A5	SPDIF1 Transmit User Buffer A5 Register	0x00000000
0x310CA3A0	SPDIF1_TX_UBUFF_B0	SPDIF1 Transmit User Buffer B0 Register	0x00000000
0x310CA3A4	SPDIF1_TX_UBUFF_B1	SPDIF1 Transmit User Buffer B1 Register	0x00000000
0x310CA3A8	SPDIF1_TX_UBUFF_B2	SPDIF1 Transmit User Buffer B2 Register	0x00000000
0x310CA3AC	SPDIF1_TX_UBUFF_B3	SPDIF1 Transmit User Buffer B3 Register	0x00000000
0x310CA3B0	SPDIF1_TX_UBUFF_B4	SPDIF1 Transmit User Buffer B4 Register	0x00000000
0x310CA3B4	SPDIF1_TX_UBUFF_B5	SPDIF1 Transmit User Buffer B5 Register	0x00000000
0x310CA3BC	SPDIF1_TX_USRUPDT	SPDIF1 User Bit Update Register	0x00000000

Table A-93: ADSP-2156x SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102E004	SPI0_CTL	SPI0 Control Register	0x00000050
0x3102E008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x3102E00C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000

Table A-93: ADSP-2156x SPI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102E010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x3102E014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x3102E018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x3102E01C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x3102E020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x3102E024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x3102E028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x3102E030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x3102E034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x3102E038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x3102E040	SPI0_STAT	SPI0 Status Register	0x00440001
0x3102E044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x3102E048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x3102E050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000
0x3102E058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000

Table A-94: ADSP-2156x SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102F004	SPI1_CTL	SPI1 Control Register	0x00000050
0x3102F008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x3102F00C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x3102F010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x3102F014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x3102F018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x3102F01C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000
0x3102F020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x3102F024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x3102F028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x3102F030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x3102F034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000

Table A-94: ADSP-2156x SPI1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3102F038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x3102F040	SPI1_STAT	SPI1 Status Register	0x00440001
0x3102F044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x3102F048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x3102F050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x3102F058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000

Table A-95: ADSP-2156x SPI2 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31030004	SPI2_CTL	SPI2 Control Register	0x00000050
0x31030008	SPI2_RXCTL	SPI2 Receive Control Register	0x00000000
0x3103000C	SPI2_TXCTL	SPI2 Transmit Control Register	0x00000000
0x31030010	SPI2_CLK	SPI2 Clock Rate Register	0x00000000
0x31030014	SPI2_DLY	SPI2 Delay Register	0x00000301
0x31030018	SPI2_SLVSEL	SPI2 Slave Select Register	0x0000FE00
0x3103001C	SPI2_RWC	SPI2 Received Word Count Register	0x00000000
0x31030020	SPI2_RWCR	SPI2 Received Word Count Reload Register	0x00000000
0x31030024	SPI2_TWC	SPI2 Transmitted Word Count Register	0x00000000
0x31030028	SPI2_TWCR	SPI2 Transmitted Word Count Reload Register	0x00000000
0x31030030	SPI2_IMSK	SPI2 Interrupt Mask Register	0x00000000
0x31030034	SPI2_IMSK_CLR	SPI2 Interrupt Mask Clear Register	0x00000000
0x31030038	SPI2_IMSK_SET	SPI2 Interrupt Mask Set Register	0x00000000
0x31030040	SPI2_STAT	SPI2 Status Register	0x00440001
0x31030044	SPI2_ILAT	SPI2 Masked Interrupt Condition Register	0x00000000
0x31030048	SPI2_ILAT_CLR	SPI2 Masked Interrupt Clear Register	0x00000000
0x31030050	SPI2_RFIFO	SPI2 Receive FIFO Data Register	0x00000000
0x31030058	SPI2_TFIFO	SPI2 Transmit FIFO Data Register	0x00000000
0x31030060	SPI2_MMRDH	SPI2 Memory Mapped Read Header	0x00000000
0x31030064	SPI2_MMTOP	SPI2 SPI Memory Top Address	0x00000000

Table A-96: ADSP-2156x SCB5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30400000	SCB5_REMAP	SCB5 Remap	0x00000000
0x30401FD0	SCB5_PERIPH_ID_4	SCB5 Periph Id 4	0x00000004
0x30401FD4	SCB5_PERIPH_ID_5	SCB5 Periph Id 5	0x00000000
0x30401FD8	SCB5_PERIPH_ID_6	SCB5 Periph Id 6	0x00000000
0x30401FDC	SCB5_PERIPH_ID_7	SCB5 Periph Id 7	0x00000000
0x30401FE0	SCB5_PERIPH_ID_0	SCB5 Periph Id 0	0x00000001
0x30401FE4	SCB5_PERIPH_ID_1	SCB5 Periph Id 1	0x000000B3
0x30401FE8	SCB5_PERIPH_ID_2	SCB5 Periph Id 2	0x0000005B
0x30401FEC	SCB5_PERIPH_ID_3	SCB5 Periph Id 3	0x00000000
0x30401FF0	SCB5_COMP_ID_0	SCB5 Comp Id 0	0x0000000D
0x30401FF4	SCB5_COMP_ID_1	SCB5 Comp Id 1	0x000000F0
0x30401FF8	SCB5_COMP_ID_2	SCB5 Comp Id 2	0x00000005
0x30401FFC	SCB5_COMP_ID_3	SCB5 Comp Id 3	0x000000B1
0x30403008	SCB5_SPI2_S_IB_FN_MOD_ISS_BM	SCB5 Spi2 S Ib.fn Mod Iss Bm	0x00000000
0x30403108	SCB5_SPI2_S_IB_FN_MOD	SCB5 Spi2 S Ib.fn Mod	0x00000000
0x30442108	SCB5_SPIF_M_FN_MOD	SCB5 Spif M.fn Mod	0x00000000

Table A-97: ADSP-2156x SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x31002004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x31002008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100200C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000

Table A-97: ADSP-2156x SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x31002040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100204C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x31002084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x31002088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100208C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310020A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x310020A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310020A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x310020C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310020C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310020C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310020CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-98: ADSP-2156x SPORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002100	SPORT1_CTL_A	SPORT1 Half SPORT 'A' Control Register	0x00000000
0x31002104	SPORT1_DIV_A	SPORT1 Half SPORT 'A' Divisor Register	0x00000000
0x31002108	SPORT1_MCTL_A	SPORT1 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100210C	SPORT1_CS0_A	SPORT1 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000

Table A-98: ADSP-2156x SPORT1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31002110	SPORT1_CS1_A	SPORT1 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002114	SPORT1_CS2_A	SPORT1 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002118	SPORT1_CS3_A	SPORT1 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002120	SPORT1_ERR_A	SPORT1 Half SPORT 'A' Error Register	0x00000000
0x31002124	SPORT1_MSTAT_A	SPORT1 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002128	SPORT1_CTL2_A	SPORT1 Half SPORT 'A' Control 2 Register	0x00000000
0x31002140	SPORT1_TXPRI_A	SPORT1 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002144	SPORT1_RXPRI_A	SPORT1 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002148	SPORT1_TXSEC_A	SPORT1 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100214C	SPORT1_RXSEC_A	SPORT1 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002180	SPORT1_CTL_B	SPORT1 Half SPORT 'B' Control Register	0x00000000
0x31002184	SPORT1_DIV_B	SPORT1 Half SPORT 'B' Divisor Register	0x00000000
0x31002188	SPORT1_MCTL_B	SPORT1 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100218C	SPORT1_CS0_B	SPORT1 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002190	SPORT1_CS1_B	SPORT1 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002194	SPORT1_CS2_B	SPORT1 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002198	SPORT1_CS3_B	SPORT1 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310021A0	SPORT1_ERR_B	SPORT1 Half SPORT 'B' Error Register	0x00000000
0x310021A4	SPORT1_MSTAT_B	SPORT1 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310021A8	SPORT1_CTL2_B	SPORT1 Half SPORT 'B' Control 2 Register	0x00000000
0x310021C0	SPORT1_TXPRI_B	SPORT1 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310021C4	SPORT1_RXPRI_B	SPORT1 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310021C8	SPORT1_TXSEC_B	SPORT1 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310021CC	SPORT1_RXSEC_B	SPORT1 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-99: ADSP-2156x SPORT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002200	SPORT2_CTL_A	SPORT2 Half SPORT 'A' Control Register	0x00000000
0x31002204	SPORT2_DIV_A	SPORT2 Half SPORT 'A' Divisor Register	0x00000000
0x31002208	SPORT2_MCTL_A	SPORT2 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100220C	SPORT2_CS0_A	SPORT2 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002210	SPORT2_CS1_A	SPORT2 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002214	SPORT2_CS2_A	SPORT2 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002218	SPORT2_CS3_A	SPORT2 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002220	SPORT2_ERR_A	SPORT2 Half SPORT 'A' Error Register	0x00000000
0x31002224	SPORT2_MSTAT_A	SPORT2 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002228	SPORT2_CTL2_A	SPORT2 Half SPORT 'A' Control 2 Register	0x00000000
0x31002240	SPORT2_TXPRI_A	SPORT2 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002244	SPORT2_RXPRI_A	SPORT2 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002248	SPORT2_TXSEC_A	SPORT2 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100224C	SPORT2_RXSEC_A	SPORT2 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002280	SPORT2_CTL_B	SPORT2 Half SPORT 'B' Control Register	0x00000000
0x31002284	SPORT2_DIV_B	SPORT2 Half SPORT 'B' Divisor Register	0x00000000
0x31002288	SPORT2_MCTL_B	SPORT2 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100228C	SPORT2_CS0_B	SPORT2 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002290	SPORT2_CS1_B	SPORT2 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002294	SPORT2_CS2_B	SPORT2 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002298	SPORT2_CS3_B	SPORT2 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310022A0	SPORT2_ERR_B	SPORT2 Half SPORT 'B' Error Register	0x00000000
0x310022A4	SPORT2_MSTAT_B	SPORT2 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310022A8	SPORT2_CTL2_B	SPORT2 Half SPORT 'B' Control 2 Register	0x00000000
0x310022C0	SPORT2_TXPRI_B	SPORT2 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000

Table A-99: ADSP-2156x SPORT2 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310022C4	SPORT2_RXPRI_B	SPORT2 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310022C8	SPORT2_TXSEC_B	SPORT2 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310022CC	SPORT2_RXSEC_B	SPORT2 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-100: ADSP-2156x SPORT3 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31002300	SPORT3_CTL_A	SPORT3 Half SPORT 'A' Control Register	0x00000000
0x31002304	SPORT3_DIV_A	SPORT3 Half SPORT 'A' Divisor Register	0x00000000
0x31002308	SPORT3_MCTL_A	SPORT3 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100230C	SPORT3_CS0_A	SPORT3 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002310	SPORT3_CS1_A	SPORT3 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002314	SPORT3_CS2_A	SPORT3 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002318	SPORT3_CS3_A	SPORT3 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002320	SPORT3_ERR_A	SPORT3 Half SPORT 'A' Error Register	0x00000000
0x31002324	SPORT3_MSTAT_A	SPORT3 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002328	SPORT3_CTL2_A	SPORT3 Half SPORT 'A' Control 2 Register	0x00000000
0x31002340	SPORT3_TXPRI_A	SPORT3 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002344	SPORT3_RXPRI_A	SPORT3 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002348	SPORT3_TXSEC_A	SPORT3 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100234C	SPORT3_RXSEC_A	SPORT3 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002380	SPORT3_CTL_B	SPORT3 Half SPORT 'B' Control Register	0x00000000
0x31002384	SPORT3_DIV_B	SPORT3 Half SPORT 'B' Divisor Register	0x00000000
0x31002388	SPORT3_MCTL_B	SPORT3 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100238C	SPORT3_CS0_B	SPORT3 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002390	SPORT3_CS1_B	SPORT3 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002394	SPORT3_CS2_B	SPORT3 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000

Table A-100: ADSP-2156x SPORT3 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31002398	SPORT3_CS3_B	SPORT3 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310023A0	SPORT3_ERR_B	SPORT3 Half SPORT 'B' Error Register	0x00000000
0x310023A4	SPORT3_MSTAT_B	SPORT3 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310023A8	SPORT3_CTL2_B	SPORT3 Half SPORT 'B' Control 2 Register	0x00000000
0x310023C0	SPORT3_TXPRI_B	SPORT3 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310023C4	SPORT3_RXPRI_B	SPORT3 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310023C8	SPORT3_TXSEC_B	SPORT3 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310023CC	SPORT3_RXSEC_B	SPORT3 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-101: ADSP-2156x SPORT4 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31002400	SPORT4_CTL_A	SPORT4 Half SPORT 'A' Control Register	0x00000000
0x31002404	SPORT4_DIV_A	SPORT4 Half SPORT 'A' Divisor Register	0x00000000
0x31002408	SPORT4_MCTL_A	SPORT4 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100240C	SPORT4_CS0_A	SPORT4 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002410	SPORT4_CS1_A	SPORT4 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002414	SPORT4_CS2_A	SPORT4 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002418	SPORT4_CS3_A	SPORT4 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002420	SPORT4_ERR_A	SPORT4 Half SPORT 'A' Error Register	0x00000000
0x31002424	SPORT4_MSTAT_A	SPORT4 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002428	SPORT4_CTL2_A	SPORT4 Half SPORT 'A' Control 2 Register	0x00000000
0x31002440	SPORT4_TXPRI_A	SPORT4 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002444	SPORT4_RXPRI_A	SPORT4 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002448	SPORT4_TXSEC_A	SPORT4 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100244C	SPORT4_RXSEC_A	SPORT4 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002480	SPORT4_CTL_B	SPORT4 Half SPORT 'B' Control Register	0x00000000
0x31002484	SPORT4_DIV_B	SPORT4 Half SPORT 'B' Divisor Register	0x00000000

Table A-101: ADSP-2156x SPORT4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002488	SPORT4_MCTL_B	SPORT4 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100248C	SPORT4_CS0_B	SPORT4 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002490	SPORT4_CS1_B	SPORT4 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002494	SPORT4_CS2_B	SPORT4 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002498	SPORT4_CS3_B	SPORT4 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310024A0	SPORT4_ERR_B	SPORT4 Half SPORT 'B' Error Register	0x00000000
0x310024A4	SPORT4_MSTAT_B	SPORT4 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310024A8	SPORT4_CTL2_B	SPORT4 Half SPORT 'B' Control 2 Register	0x00000000
0x310024C0	SPORT4_TXPRI_B	SPORT4 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310024C4	SPORT4_RXPRI_B	SPORT4 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310024C8	SPORT4_TXSEC_B	SPORT4 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310024CC	SPORT4_RXSEC_B	SPORT4 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-102: ADSP-2156x SPORT5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002500	SPORT5_CTL_A	SPORT5 Half SPORT 'A' Control Register	0x00000000
0x31002504	SPORT5_DIV_A	SPORT5 Half SPORT 'A' Divisor Register	0x00000000
0x31002508	SPORT5_MCTL_A	SPORT5 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100250C	SPORT5_CS0_A	SPORT5 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002510	SPORT5_CS1_A	SPORT5 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002514	SPORT5_CS2_A	SPORT5 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002518	SPORT5_CS3_A	SPORT5 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002520	SPORT5_ERR_A	SPORT5 Half SPORT 'A' Error Register	0x00000000
0x31002524	SPORT5_MSTAT_A	SPORT5 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002528	SPORT5_CTL2_A	SPORT5 Half SPORT 'A' Control 2 Register	0x00000000

Table A-102: ADSP-2156x SPORT5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002540	SPORT5_TXPRI_A	SPORT5 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002544	SPORT5_RXPRI_A	SPORT5 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002548	SPORT5_TXSEC_A	SPORT5 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100254C	SPORT5_RXSEC_A	SPORT5 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002580	SPORT5_CTL_B	SPORT5 Half SPORT 'B' Control Register	0x00000000
0x31002584	SPORT5_DIV_B	SPORT5 Half SPORT 'B' Divisor Register	0x00000000
0x31002588	SPORT5_MCTL_B	SPORT5 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100258C	SPORT5_CS0_B	SPORT5 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002590	SPORT5_CS1_B	SPORT5 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002594	SPORT5_CS2_B	SPORT5 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002598	SPORT5_CS3_B	SPORT5 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310025A0	SPORT5_ERR_B	SPORT5 Half SPORT 'B' Error Register	0x00000000
0x310025A4	SPORT5_MSTAT_B	SPORT5 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310025A8	SPORT5_CTL2_B	SPORT5 Half SPORT 'B' Control 2 Register	0x00000000
0x310025C0	SPORT5_TXPRI_B	SPORT5 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310025C4	SPORT5_RXPRI_B	SPORT5 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310025C8	SPORT5_TXSEC_B	SPORT5 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310025CC	SPORT5_RXSEC_B	SPORT5 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-103: ADSP-2156x SPORT6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002600	SPORT6_CTL_A	SPORT6 Half SPORT 'A' Control Register	0x00000000
0x31002604	SPORT6_DIV_A	SPORT6 Half SPORT 'A' Divisor Register	0x00000000
0x31002608	SPORT6_MCTL_A	SPORT6 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100260C	SPORT6_CS0_A	SPORT6 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002610	SPORT6_CS1_A	SPORT6 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000

Table A-103: ADSP-2156x SPORT6 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31002614	SPORT6_CS2_A	SPORT6 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002618	SPORT6_CS3_A	SPORT6 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002620	SPORT6_ERR_A	SPORT6 Half SPORT 'A' Error Register	0x00000000
0x31002624	SPORT6_MSTAT_A	SPORT6 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002628	SPORT6_CTL2_A	SPORT6 Half SPORT 'A' Control 2 Register	0x00000000
0x31002640	SPORT6_TXPRI_A	SPORT6 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002644	SPORT6_RXPRI_A	SPORT6 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002648	SPORT6_TXSEC_A	SPORT6 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100264C	SPORT6_RXSEC_A	SPORT6 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002680	SPORT6_CTL_B	SPORT6 Half SPORT 'B' Control Register	0x00000000
0x31002684	SPORT6_DIV_B	SPORT6 Half SPORT 'B' Divisor Register	0x00000000
0x31002688	SPORT6_MCTL_B	SPORT6 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100268C	SPORT6_CS0_B	SPORT6 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002690	SPORT6_CS1_B	SPORT6 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002694	SPORT6_CS2_B	SPORT6 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002698	SPORT6_CS3_B	SPORT6 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310026A0	SPORT6_ERR_B	SPORT6 Half SPORT 'B' Error Register	0x00000000
0x310026A4	SPORT6_MSTAT_B	SPORT6 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310026A8	SPORT6_CTL2_B	SPORT6 Half SPORT 'B' Control 2 Register	0x00000000
0x310026C0	SPORT6_TXPRI_B	SPORT6 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310026C4	SPORT6_RXPRI_B	SPORT6 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310026C8	SPORT6_TXSEC_B	SPORT6 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310026CC	SPORT6_RXSEC_B	SPORT6 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-104: ADSP-2156x SPORT7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002700	SPORT7_CTL_A	SPORT7 Half SPORT 'A' Control Register	0x00000000
0x31002704	SPORT7_DIV_A	SPORT7 Half SPORT 'A' Divisor Register	0x00000000
0x31002708	SPORT7_MCTL_A	SPORT7 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100270C	SPORT7_CS0_A	SPORT7 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002710	SPORT7_CS1_A	SPORT7 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002714	SPORT7_CS2_A	SPORT7 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002718	SPORT7_CS3_A	SPORT7 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002720	SPORT7_ERR_A	SPORT7 Half SPORT 'A' Error Register	0x00000000
0x31002724	SPORT7_MSTAT_A	SPORT7 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002728	SPORT7_CTL2_A	SPORT7 Half SPORT 'A' Control 2 Register	0x00000000
0x31002740	SPORT7_TXPRI_A	SPORT7 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002744	SPORT7_RXPRI_A	SPORT7 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002748	SPORT7_TXSEC_A	SPORT7 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100274C	SPORT7_RXSEC_A	SPORT7 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002780	SPORT7_CTL_B	SPORT7 Half SPORT 'B' Control Register	0x00000000
0x31002784	SPORT7_DIV_B	SPORT7 Half SPORT 'B' Divisor Register	0x00000000
0x31002788	SPORT7_MCTL_B	SPORT7 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100278C	SPORT7_CS0_B	SPORT7 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002790	SPORT7_CS1_B	SPORT7 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002794	SPORT7_CS2_B	SPORT7 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002798	SPORT7_CS3_B	SPORT7 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310027A0	SPORT7_ERR_B	SPORT7 Half SPORT 'B' Error Register	0x00000000
0x310027A4	SPORT7_MSTAT_B	SPORT7 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310027A8	SPORT7_CTL2_B	SPORT7 Half SPORT 'B' Control 2 Register	0x00000000
0x310027C0	SPORT7_TXPRI_B	SPORT7 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000

Table A-104: ADSP-2156x SPORT7 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310027C4	SPORT7_RXPRI_B	SPORT7 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310027C8	SPORT7_TXSEC_B	SPORT7 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310027CC	SPORT7_RXSEC_B	SPORT7 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-105: ADSP-2156x SPU0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108B000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x3108B004	SPU0_STAT	SPU0 Status Register	0x00000000
0x3108B400	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B404	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B408	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B40C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B410	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B414	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B418	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B41C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B420	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B424	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B428	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B42C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B430	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B434	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B438	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B43C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B440	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B444	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B448	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B44C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B450	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B454	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108B458	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B45C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B460	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B464	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B468	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B46C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B470	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B474	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B478	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B47C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B480	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B484	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B488	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B48C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B490	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B494	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B498	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B49C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108B4D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4F4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4F8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B4FC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B500	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B504	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B508	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B50C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B510	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B514	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B518	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B51C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B520	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B524	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B528	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B52C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B530	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B534	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B538	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B53C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B540	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B544	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B548	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B54C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108B550	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B554	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B558	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B55C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B560	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B564	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B568	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B56C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B570	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B574	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B578	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B57C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B580	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B584	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B588	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B58C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B590	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B594	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B598	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B59C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108B5CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5F4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5F8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B5FC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B600	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B604	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B608	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B60C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B610	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B614	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B618	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B61C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B620	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108B840	SPU0_SECURECTL	SPU0 Secure Control Register	0x00000000
0x3108B84C	SPU0_SECURECHK	SPU0 Secure Check Register	0xFFFFFFFF
0x3108B980	SPU0_SECUREC[n]	SPU0 Secure Core Registers	0x00000001
0x3108BA00	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA04	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA08	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA0C	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA10	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA14	SPU0_SECUREEP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108BA18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108BA94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BA9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BABC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BACC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BADC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAF4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAF8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BAFC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108BB10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108BB8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BB9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBBC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBCC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBDC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBF4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBF8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BBFC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-105: ADSP-2156x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108BC08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108BC20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-106: ADSP-2156x SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31093000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x31093004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x31093010	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31093014	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31093018	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109301C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31093020	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31093024	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31093028	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109302C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31093030	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31093034	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31093038	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109303C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31093040	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31093044	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31093048	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109304C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31093050	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31093054	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000

Table A-106: ADSP-2156x SWU1 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31093058	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109305C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31093060	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31093064	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31093068	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109306C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31093070	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31093074	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31093078	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109307C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31093080	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31093084	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31093088	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109308C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000

Table A-107: ADSP-2156x SWU11 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31097000	SWU11_GCTL	SWU11 Global Control Register	0x00000000
0x31097004	SWU11_GSTAT	SWU11 Global Status Register	0x00000000
0x31097010	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097014	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097018	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109701C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097020	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097024	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097028	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109702C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097030	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097034	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097038	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000

Table A-107: ADSP-2156x SWU11 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109703C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097040	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097044	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097048	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109704C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097050	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097054	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097058	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109705C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097060	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097064	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097068	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109706C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097070	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097074	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097078	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109707C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097080	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097084	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097088	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109708C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000

Table A-108: ADSP-2156x SWU12 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A8000	SWU12_GCTL	SWU12 Global Control Register	0x00000000
0x310A8004	SWU12_GSTAT	SWU12 Global Status Register	0x00000000
0x310A8010	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x310A8014	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x310A8018	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x310A801C	SWU12_ID[n]	SWU12 ID Register n	0x00000000

Table A-108: ADSP-2156x SWU12 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310A8020	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x310A8024	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x310A8028	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x310A802C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000
0x310A8030	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x310A8034	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x310A8038	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x310A803C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x310A8040	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x310A8044	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x310A8048	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x310A804C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000
0x310A8050	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x310A8054	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x310A8058	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x310A805C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x310A8060	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x310A8064	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x310A8068	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x310A806C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000
0x310A8070	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x310A8074	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x310A8078	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x310A807C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x310A8080	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x310A8084	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x310A8088	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x310A808C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000

Table A-109: ADSP-2156x SWU13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109E000	SWU13_GCTL	SWU13 Global Control Register	0x00000000
0x3109E004	SWU13_GSTAT	SWU13 Global Status Register	0x00000000
0x3109E010	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E014	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E018	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E01C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E020	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E024	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E028	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E02C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E030	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E034	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E038	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E03C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E040	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E044	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E048	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E04C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E050	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E054	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E058	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E05C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E060	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E064	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E068	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E06C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E070	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E074	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E078	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E07C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E080	SWU13_CNT[n]	SWU13 Count Register n	0x00000000

Table A-109: ADSP-2156x SWU13 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3109E084	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E088	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E08C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000

Table A-110: ADSP-2156x SWU2 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31094000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x31094004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x31094010	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31094014	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31094018	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109401C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31094020	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31094024	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31094028	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109402C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31094030	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31094034	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31094038	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109403C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31094040	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31094044	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31094048	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109404C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31094050	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31094054	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31094058	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109405C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31094060	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31094064	SWU2_TARG[n]	SWU2 Target Register n	0x00000000

Table A-110: ADSP-2156x SWU2 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31094068	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109406C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31094070	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31094074	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31094078	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109407C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31094080	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31094084	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31094088	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109408C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000

Table A-111: ADSP-2156x SWU7 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31140000	SWU7_GCTL	SWU7 Global Control Register	0x00000000
0x31140004	SWU7_GSTAT	SWU7 Global Status Register	0x00000000
0x31140010	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140014	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140018	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114001C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140020	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140024	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140028	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114002C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140030	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140034	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140038	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114003C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140040	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140044	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140048	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000

Table A-111: ADSP-2156x SWU7 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3114004C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140050	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140054	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140058	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114005C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140060	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140064	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140068	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114006C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140070	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140074	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140078	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114007C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140080	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140084	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140088	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114008C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000

Table A-112: ADSP-2156x SWU8 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31141000	SWU8_GCTL	SWU8 Global Control Register	0x00000000
0x31141004	SWU8_GSTAT	SWU8 Global Status Register	0x00000000
0x31141010	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141014	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141018	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114101C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141020	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141024	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141028	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114102C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000

Table A-112: ADSP-2156x SWU8 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31141030	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141034	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141038	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114103C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141040	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141044	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141048	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114104C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000
0x31141050	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141054	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141058	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114105C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141060	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141064	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141068	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114106C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000
0x31141070	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141074	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141078	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114107C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141080	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141084	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141088	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114108C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000

Table A-113: ADSP-2156x TAPC MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31130000	TAPC_IDCODE	TAPC IDCODE Register	0x028200CB
0x31130004	TAPC_USERCODE	TAPC USERCODE Register	0x00000000
0x31130008	TAPC_SDBGKEY_CTL	TAPC Secure Debug Key Control Register	0x00000000

Table A-113: ADSP-2156x TAPC MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3113000C	TAPC_SDBGKEY_STAT	TAPC Secure Debug Key Status Register	0x00000000
0x31130010	TAPC_SDBGKEY0	TAPC Secure Debug Key 0 Register	0x00000000
0x31130014	TAPC_SDBGKEY1	TAPC Secure Debug Key 1 Register	0x00000000
0x31130018	TAPC_SDBGKEY2	TAPC Secure Debug Key 2 Register	0x00000000
0x3113001C	TAPC_SDBGKEY3	TAPC Secure Debug Key 3 Register	0x00000000
0x31131000	TAPC_DBGCTL	TAPC Debug Control Register	0x00000000

Table A-114: ADSP-2156x TIMER0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31018004	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x31018008	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x3101800C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x31018010	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x31018014	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000
0x31018018	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x3101801C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000003FF
0x31018020	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000003FF
0x31018024	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000003FF
0x31018028	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x3101802C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x31018030	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x31018034	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x31018038	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x3101803C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000
0x31018040	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x31018060	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018064	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018068	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101806C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018070	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-114: ADSP-2156x TIMER0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31018080	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018084	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018088	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101808C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018090	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310180A0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310180A4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310180A8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310180AC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310180B0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310180C0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310180C4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310180C8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310180CC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310180D0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310180E0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310180E4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310180E8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310180EC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310180F0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31018100	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018104	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018108	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101810C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018110	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31018120	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018124	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018128	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101812C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018130	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31018140	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000

Table A-114: ADSP-2156x TIMER0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x31018144	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018148	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101814C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018150	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31018160	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018164	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018168	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101816C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018170	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31018180	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31018184	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31018188	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3101818C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31018190	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-115: ADSP-2156x TMU0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31016800	TMU0_CTL	TMU0 TMU Control Register	0x00000000
0x31016804	TMU0_TEMP	TMU0 Temperature Value Register	0x00000000
0x31016808	TMU0_AVG	TMU0 Averaging Register	0x00000000
0x3101680C	TMU0_FLT_LIM_HI	TMU0 Fault High Limit Register	0x000000FF
0x31016810	TMU0_ALRT_LIM_HI	TMU0 Alert High Limit Register	0x000000FF
0x31016814	TMU0_FLT_LIM_LO	TMU0 Fault Low Limit Register	0x00000100
0x31016818	TMU0_ALRT_LIM_LO	TMU0 Alert Low Limit Register	0x00000100
0x3101681C	TMU0_STAT	TMU0 Status Register	0x00000000
0x31016824	TMU0_GAIN	TMU0 Gain Value Register	0x00000000
0x31016828	TMU0_IMSK	TMU0 Interrupt Mask Register	0x0000000F
0x3101682C	TMU0_OFFSET	TMU0 Offset Register	0x00000000
0x31016834	TMU0_CNV_BLANK	TMU0 Temperature conversion blank register	0x03100000
0x31016838	TMU0_REFR_CNTR	TMU0 Temperature Refresh Counter	0x00000000

Table A-116: ADSP-2156x TRNG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D0000	TRNG0_INPUT[n]	TRNG0 TRNG Input Registers	0x00000000
0x310D0000	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0004	TRNG0_INPUT[n]	TRNG0 TRNG Input Registers	0x00000000
0x310D0004	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0008	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D000C	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0010	TRNG0_STAT	TRNG0 TRNG Status Register	0x00000000
0x310D0010	TRNG0_INTACK	TRNG0 TRNG Interrupt Acknowledge Register	0x00000000
0x310D0014	TRNG0_CTL	TRNG0 TRNG Control Register	0x00000000
0x310D0018	TRNG0_CFG	TRNG0 TRNG Configuration Register	0x00000000
0x310D001C	TRNG0_ALMCNT	TRNG0 TRNG Alarm Counter Register	0x000000FF
0x310D0020	TRNG0_FROEN	TRNG0 TRNG FRO Enable Register	0x000000FF
0x310D0024	TRNG0_FRODETUNE	TRNG0 TRNG FRO De-tune Register	0x00000000
0x310D0028	TRNG0_ALMMSK	TRNG0 TRNG Alarm Mask Register	0x00000000
0x310D002C	TRNG0_ALMSTP	TRNG0 TRNG Alarm Stop Register	0x00000000
0x310D0030	TRNG0_LFSR_L	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D0034	TRNG0_LFSR_M	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D0038	TRNG0_LFSR_H	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D003C	TRNG0_CNT	TRNG0 Counter Register	0x00000000
0x310D0040	TRNG0_RUNCNT	TRNG0 TRNG Run Count Registers	0x00000000
0x310D0040	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x310D0044	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0044	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x310D0048	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0048	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x310D004C	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D004C	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x310D0050	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0050	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x310D0054	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0054	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000

Table A-116: ADSP-2156x TRNG0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x310D0058	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D005C	TRNG0_MONOBITCNT	TRNG0 TRNG Monobit Test Result Register	0x00002710
0x310D0060	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0060	TRNG0_V[n]	TRNG0 TRNG Post-Process "V" Value Registers	0x00000000
0x310D0064	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0064	TRNG0_V[n]	TRNG0 TRNG Post-Process "V" Value Registers	0x00000000
0x310D0068	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D006C	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0070	TRNG0_TEST	TRNG0 TRNG Test Register	0x00000000
0x310D0074	TRNG0_BLKCNT	TRNG0 TRNG Block Count Register	0x00000000

Table A-117: ADSP-2156x TRU0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108A000	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A004	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A008	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A00C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A010	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A014	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A018	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A01C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A020	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A024	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A028	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A02C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A030	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A034	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A038	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A03C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A040	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-117: ADSP-2156x TRU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108A044	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A048	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A04C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A050	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A054	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A058	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A05C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A060	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A064	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A068	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A06C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A070	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A074	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A078	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A07C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A080	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A084	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A088	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A08C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A090	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A094	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A098	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A09C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0BC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-117: ADSP-2156x TRU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108A0C0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0C4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0C8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0CC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0DC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0EC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0FC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A100	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A104	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A108	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A10C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A110	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A114	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A118	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A11C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A120	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A124	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A128	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A12C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A130	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A134	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A138	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-117: ADSP-2156x TRU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108A13C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A140	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A144	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A148	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A14C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A150	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A154	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A158	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A15C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A160	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A164	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A168	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A16C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A170	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A174	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A178	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A17C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A180	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A184	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A188	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A18C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A190	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A194	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A198	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A19C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-117: ADSP-2156x TRU0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3108A1B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1BC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1CC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1D0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1D4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A7E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x3108A7E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x3108A7EC	TRU0_STAT	TRU0 Status Information Register	0x00000000
0x3108A7F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table A-118: ADSP-2156x TWI0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31001400	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x31001404	TWI0_CTL	TWI0 Control Register	0x00000000
0x31001408	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x3100140C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x31001410	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000
0x31001414	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x31001418	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000
0x3100141C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x31001420	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x31001424	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x31001428	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x3100142C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000
0x31001480	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x31001484	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x31001488	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000

Table A-118: ADSP-2156x TWI0 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3100148C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table A-119: ADSP-2156x TWI1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31001500	TWI1_CLKDIV	TWI1 SCL Clock Divider Register	0x00000000
0x31001504	TWI1_CTL	TWI1 Control Register	0x00000000
0x31001508	TWI1_SLVCTL	TWI1 Slave Mode Control Register	0x00000000
0x3100150C	TWI1_SLVSTAT	TWI1 Slave Mode Status Register	0x00000000
0x31001510	TWI1_SLVADDR	TWI1 Slave Mode Address Register	0x00000000
0x31001514	TWI1_MSTRCTL	TWI1 Master Mode Control Registers	0x00000000
0x31001518	TWI1_MSTRSTAT	TWI1 Master Mode Status Register	0x00000000
0x3100151C	TWI1_MSTRADDR	TWI1 Master Mode Address Register	0x00000000
0x31001520	TWI1_ISTAT	TWI1 Interrupt Status Register	0x00000000
0x31001524	TWI1_IMSK	TWI1 Interrupt Mask Register	0x00000000
0x31001528	TWI1_FIFOCTL	TWI1 FIFO Control Register	0x00000000
0x3100152C	TWI1_FIFOSTAT	TWI1 FIFO Status Register	0x00000000
0x31001580	TWI1_TXDATA8	TWI1 Tx Data Single-Byte Register	0x00000000
0x31001584	TWI1_TXDATA16	TWI1 Tx Data Double-Byte Register	0x00000000
0x31001588	TWI1_RXDATA8	TWI1 Rx Data Single-Byte Register	0x00000000
0x3100158C	TWI1_RXDATA16	TWI1 Rx Data Double-Byte Register	0x00000000

Table A-120: ADSP-2156x TWI2 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31001600	TWI2_CLKDIV	TWI2 SCL Clock Divider Register	0x00000000
0x31001604	TWI2_CTL	TWI2 Control Register	0x00000000
0x31001608	TWI2_SLVCTL	TWI2 Slave Mode Control Register	0x00000000
0x3100160C	TWI2_SLVSTAT	TWI2 Slave Mode Status Register	0x00000000
0x31001610	TWI2_SLVADDR	TWI2 Slave Mode Address Register	0x00000000
0x31001614	TWI2_MSTRCTL	TWI2 Master Mode Control Registers	0x00000000
0x31001618	TWI2_MSTRSTAT	TWI2 Master Mode Status Register	0x00000000

Table A-120: ADSP-2156x TWI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100161C	TWI2_MSTRADDR	TWI2 Master Mode Address Register	0x00000000
0x31001620	TWI2_ISTAT	TWI2 Interrupt Status Register	0x00000000
0x31001624	TWI2_IMSK	TWI2 Interrupt Mask Register	0x00000000
0x31001628	TWI2_FIFOCTL	TWI2 FIFO Control Register	0x00000000
0x3100162C	TWI2_FIFOSTAT	TWI2 FIFO Status Register	0x00000000
0x31001680	TWI2_TXDATA8	TWI2 Tx Data Single-Byte Register	0x00000000
0x31001684	TWI2_TXDATA16	TWI2 Tx Data Double-Byte Register	0x00000000
0x31001688	TWI2_RXDATA8	TWI2 Rx Data Single-Byte Register	0x00000000
0x3100168C	TWI2_RXDATA16	TWI2 Rx Data Double-Byte Register	0x00000000

Table A-121: ADSP-2156x TWI3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001000	TWI3_CLKDIV	TWI3 SCL Clock Divider Register	0x00000000
0x31001004	TWI3_CTL	TWI3 Control Register	0x00000000
0x31001008	TWI3_SLVCTL	TWI3 Slave Mode Control Register	0x00000000
0x3100100C	TWI3_SLVSTAT	TWI3 Slave Mode Status Register	0x00000000
0x31001010	TWI3_SLVADDR	TWI3 Slave Mode Address Register	0x00000000
0x31001014	TWI3_MSTRCTL	TWI3 Master Mode Control Registers	0x00000000
0x31001018	TWI3_MSTRSTAT	TWI3 Master Mode Status Register	0x00000000
0x3100101C	TWI3_MSTRADDR	TWI3 Master Mode Address Register	0x00000000
0x31001020	TWI3_ISTAT	TWI3 Interrupt Status Register	0x00000000
0x31001024	TWI3_IMSK	TWI3 Interrupt Mask Register	0x00000000
0x31001028	TWI3_FIFOCTL	TWI3 FIFO Control Register	0x00000000
0x3100102C	TWI3_FIFOSTAT	TWI3 FIFO Status Register	0x00000000
0x31001080	TWI3_TXDATA8	TWI3 Tx Data Single-Byte Register	0x00000000
0x31001084	TWI3_TXDATA16	TWI3 Tx Data Double-Byte Register	0x00000000
0x31001088	TWI3_RXDATA8	TWI3 Rx Data Single-Byte Register	0x00000000
0x3100108C	TWI3_RXDATA16	TWI3 Rx Data Double-Byte Register	0x00000000

Table A-122: ADSP-2156x TWI4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001100	TWI4_CLKDIV	TWI4 SCL Clock Divider Register	0x00000000
0x31001104	TWI4_CTL	TWI4 Control Register	0x00000000
0x31001108	TWI4_SLVCTL	TWI4 Slave Mode Control Register	0x00000000
0x3100110C	TWI4_SLVSTAT	TWI4 Slave Mode Status Register	0x00000000
0x31001110	TWI4_SLVADDR	TWI4 Slave Mode Address Register	0x00000000
0x31001114	TWI4_MSTRCTL	TWI4 Master Mode Control Registers	0x00000000
0x31001118	TWI4_MSTRSTAT	TWI4 Master Mode Status Register	0x00000000
0x3100111C	TWI4_MSTRADDR	TWI4 Master Mode Address Register	0x00000000
0x31001120	TWI4_ISTAT	TWI4 Interrupt Status Register	0x00000000
0x31001124	TWI4_IMSK	TWI4 Interrupt Mask Register	0x00000000
0x31001128	TWI4_FIFOCTL	TWI4 FIFO Control Register	0x00000000
0x3100112C	TWI4_FIFOSTAT	TWI4 FIFO Status Register	0x00000000
0x31001180	TWI4_TXDATA8	TWI4 Tx Data Single-Byte Register	0x00000000
0x31001184	TWI4_TXDATA16	TWI4 Tx Data Double-Byte Register	0x00000000
0x31001188	TWI4_RXDATA8	TWI4 Rx Data Single-Byte Register	0x00000000
0x3100118C	TWI4_RXDATA16	TWI4 Rx Data Double-Byte Register	0x00000000

Table A-123: ADSP-2156x TWI5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001200	TWI5_CLKDIV	TWI5 SCL Clock Divider Register	0x00000000
0x31001204	TWI5_CTL	TWI5 Control Register	0x00000000
0x31001208	TWI5_SLVCTL	TWI5 Slave Mode Control Register	0x00000000
0x3100120C	TWI5_SLVSTAT	TWI5 Slave Mode Status Register	0x00000000
0x31001210	TWI5_SLVADDR	TWI5 Slave Mode Address Register	0x00000000
0x31001214	TWI5_MSTRCTL	TWI5 Master Mode Control Registers	0x00000000
0x31001218	TWI5_MSTRSTAT	TWI5 Master Mode Status Register	0x00000000
0x3100121C	TWI5_MSTRADDR	TWI5 Master Mode Address Register	0x00000000
0x31001220	TWI5_ISTAT	TWI5 Interrupt Status Register	0x00000000
0x31001224	TWI5_IMSK	TWI5 Interrupt Mask Register	0x00000000
0x31001228	TWI5_FIFOCTL	TWI5 FIFO Control Register	0x00000000

Table A-123: ADSP-2156x TWI5 MMR Register Addresses (Continued)

Memory Map-ped Address	Register Name	Description	Reset Value
0x3100122C	TWI5_FIFOSTAT	TWI5 FIFO Status Register	0x00000000
0x31001280	TWI5_TXDATA8	TWI5 Tx Data Single-Byte Register	0x00000000
0x31001284	TWI5_TXDATA16	TWI5 Tx Data Double-Byte Register	0x00000000
0x31001288	TWI5_RXDATA8	TWI5 Rx Data Single-Byte Register	0x00000000
0x3100128C	TWI5_RXDATA16	TWI5 Rx Data Double-Byte Register	0x00000000

Table A-124: ADSP-2156x UART0 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31003004	UART0_CTL	UART0 Control Register	0x00000000
0x31003008	UART0_STAT	UART0 Status Register	0x000000A0
0x3100300C	UART0_SCR	UART0 Scratch Register	0x00000000
0x31003010	UART0_CLK	UART0 Clock Rate Register	0x0000FFFF
0x31003014	UART0_IMSK	UART0 Interrupt Mask Register	0x00000000
0x31003018	UART0_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x3100301C	UART0_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x31003020	UART0_RBR	UART0 Receive Buffer Register	0x00000000
0x31003024	UART0_THR	UART0 Transmit Hold Register	0x00000000
0x31003028	UART0_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x3100302C	UART0_TSR	UART0 Transmit Shift Register	0x000007FF
0x31003030	UART0_RSR	UART0 Receive Shift Register	0x00000000
0x31003034	UART0_TXCNT	UART0 Transmit Counter Register	0x00000000
0x31003038	UART0_RXCNT	UART0 Receive Counter Register	0x00000000

Table A-125: ADSP-2156x UART1 MMR Register Addresses

Memory Map-ped Address	Register Name	Description	Reset Value
0x31003404	UART1_CTL	UART1 Control Register	0x00000000
0x31003408	UART1_STAT	UART1 Status Register	0x000000A0
0x3100340C	UART1_SCR	UART1 Scratch Register	0x00000000
0x31003410	UART1_CLK	UART1 Clock Rate Register	0x0000FFFF
0x31003414	UART1_IMSK	UART1 Interrupt Mask Register	0x00000000

Table A-125: ADSP-2156x UART1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31003418	UART1_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x3100341C	UART1_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x31003420	UART1_RBR	UART1 Receive Buffer Register	0x00000000
0x31003424	UART1_THR	UART1 Transmit Hold Register	0x00000000
0x31003428	UART1_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x3100342C	UART1_TSR	UART1 Transmit Shift Register	0x000007FF
0x31003430	UART1_RSR	UART1 Receive Shift Register	0x00000000
0x31003434	UART1_TXCNT	UART1 Transmit Counter Register	0x00000000
0x31003438	UART1_RXCNT	UART1 Receive Counter Register	0x00000000

Table A-126: ADSP-2156x UART2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31003804	UART2_CTL	UART2 Control Register	0x00000000
0x31003808	UART2_STAT	UART2 Status Register	0x000000A0
0x3100380C	UART2_SCR	UART2 Scratch Register	0x00000000
0x31003810	UART2_CLK	UART2 Clock Rate Register	0x0000FFFF
0x31003814	UART2_IMSK	UART2 Interrupt Mask Register	0x00000000
0x31003818	UART2_IMSK_SET	UART2 Interrupt Mask Set Register	0x00000000
0x3100381C	UART2_IMSK_CLR	UART2 Interrupt Mask Clear Register	0x00000000
0x31003820	UART2_RBR	UART2 Receive Buffer Register	0x00000000
0x31003824	UART2_THR	UART2 Transmit Hold Register	0x00000000
0x31003828	UART2_TAIP	UART2 Transmit Address/Insert Pulse Register	0x00000000
0x3100382C	UART2_TSR	UART2 Transmit Shift Register	0x000007FF
0x31003830	UART2_RSR	UART2 Receive Shift Register	0x00000000
0x31003834	UART2_TXCNT	UART2 Transmit Counter Register	0x00000000
0x31003838	UART2_RXCNT	UART2 Receive Counter Register	0x00000000

Table A-127: ADSP-2156x WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31008000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0

Table A-127: ADSP-2156x WDOG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31008004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x31008008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000
0x3100800C	WDOG0_WIN	WDOG0 Watchdog Timer Window Register	0xFFFFFFFF

Table A-128: ADSP-2156x WDOG1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31008800	WDOG1_CTL	WDOG1 Control Register	0x0000AD0
0x31008804	WDOG1_CNT	WDOG1 Count Register	0x00000000
0x31008808	WDOG1_STAT	WDOG1 Watchdog Timer Status Register	0x00000000
0x3100880C	WDOG1_WIN	WDOG1 Watchdog Timer Window Register	0xFFFFFFFF